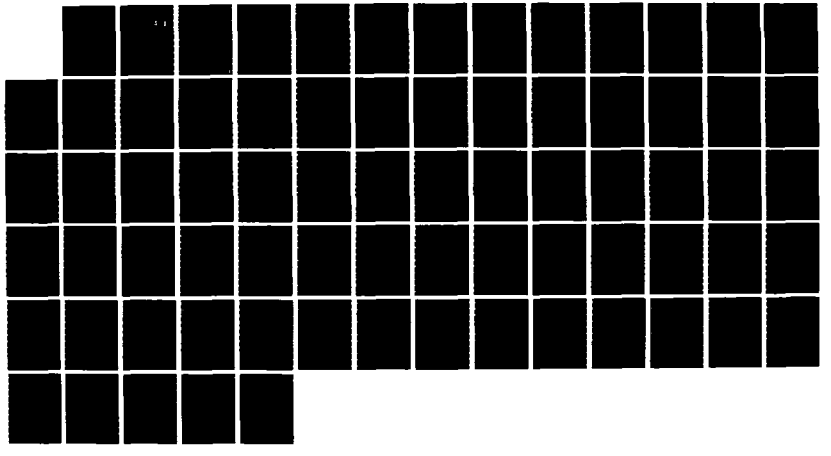
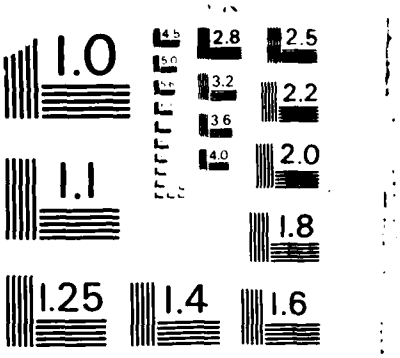
AD-A177 124 PRECEDENCE-CONSTRAINED SCHEDULING WITH MINIMUM TIME AND 1/1
COMMUNICATION(U) ILLINOIS UNIV AT URBANA COORDINATED
SCIENCE LAB A L PRASTEIN JAN 87 UILU-ENG-87-2207
UNCLASSIFIED N00014-85-K-0578 F/G 17/2 NL





MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963

January 1987

UILU-ENG-87-2207
ACT-75

COORDINATED SCIENCE LABORATORY
College of Engineering
Applied Computation Theory

PRECEDENCE-CONSTRAINED SCHEDULING WITH MINIMUM TIME AND COMMUNICATION

Marsha Lise Prastein

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG- 87-2207 (ACT-75)	5. MONITORING ORGANIZATION REPORT NUMBER(S) N/A
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois	6b. OFFICE SYMBOL (If applicable) N/A
7a. NAME OF MONITORING ORGANIZATION Office of Naval Research	
6c. ADDRESS (City, State and ZIP Code) 1101 W. Springfield Avenue Urbana, Illinois 61801	7b. ADDRESS (City, State and ZIP Code) 800 N. Quincy Street Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Joint Services Electronics Program & ONR	8b. OFFICE SYMBOL (If applicable) N/A
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER ONR: N00014-85-K-0570 JSEP: N00014-84-C-0149	
8c. ADDRESS (City, State and ZIP Code) 800 N. Quincy Street Arlington, VA 22217	10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
N/A	N/A	N/A	N/A

11. TITLE (Include Security Classification) Precedence-Constrained Scheduling with Minimum Time and Communication

12. PERSONAL AUTHOR(S)
Marsha Lise Prastein

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	January 1987	69

16. SUPPLEMENTARY NOTATION
N/A

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR	scheduling, precedence, task assignment, communication complexity, computational complexity, NP-completeness graph, tree

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

I consider task systems modeled by directed acyclic graphs in which nodes represent tasks and arcs express precedence constraints, and each task can be computed by a processor in one unit of time. It is known that if there are only two processors or if the graph is a tree, then there are polynomial time algorithms for scheduling the graph in minimum time, but in general the minimum time scheduling problem is NP-complete. The communication cost of a schedule is the number of pairs (p,x) such that processor p does not compute task x but computes an immediate successor of x; that is, the result of x must be communicated to p. I consider the problem of finding schedules that minimize finishing time and among those, finding schedules that minimize communication. I prove that the problem with two processors on an arbitrary graph is NP-complete. The problem with arbitrarily many processors on a tree is also NP-complete. The case of two processors on a tree is open in general, but I establish tight bounds for two processors on the indirected complete ternary tree of height k:

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION (Continue on back)
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS ☐	Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
		NONE

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

19. continued

for minimum time, communication $k-\log_3 k+3$ is achievable, and communication $k-\log_3 k+1$ is necessary.

PRECEDENCE-CONSTRAINED SCHEDULING WITH MINIMUM TIME AND COMMUNICATION

BY

MARSHA LISE PRASTEIN

B.S., University of Chicago, 1983

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1987

Urbana, Illinois

ABSTRACT

I consider task systems modeled by directed acyclic graphs in which nodes represent tasks and arcs express precedence constraints, and each task can be computed by a processor in one unit of time. It is known that if there are only two processors or if the graph is a tree, then there are polynomial time algorithms for scheduling the graph in minimum time, but in general the minimum time scheduling problem is NP-complete. The *communication cost* of a schedule is the number of pairs (p, x) such that processor p does not compute task x but computes an immediate successor of x; that is, the result of x must be communicated to p. I consider the problem of finding schedules that minimize finishing time and among those, finding schedules that minimize communication. I prove that the problem with two processors on an arbitrary graph is NP-complete. The problem with arbitrarily many processors on a tree is also NP-complete. The case of two processors on a tree is open in general, but I establish tight bounds for two processors on the indirected complete ternary tree of height k: for minimum time, communication $k - \log_3 k + 3$ is achievable, and communication $k - \log_3 k + 1$ is necessary.

ACKNOWLEDGMENTS

I would like to thank my advisor, Michael C. Loui, for his inspired and inspiring help and guidance, as well as his undying confidence, support, and patience. I also thank Michael Wu for all of his help with TROFF and the printer. Thanks to my departmental friends Dee, Chip M., Rick, Chip Q., Pat, and the rest of "The Gang" for helping me to maintain sanity during my thesis marathons, and thanks to the Illini Squares, particularly the C1 tape group; I needed the diversion. Special thanks also to Kevin for his love and biweekly visits that helped keep me going. I am grateful to my cats, Fargo and Daphne for keeping me warm at night and not getting too sick when I needed to work. Lastly, I would like to thank my family for supporting and encouraging me in everything I try to accomplish and letting me know that they believe in me.

Supported by The Office of Naval Research under Contract N00014-85-K-0570 and by The Joint Services Electronics Program (U.S. Army, U.S. Navy, U.S. Air Force) under Contract N00014-85-C-0149.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION/LITERATURE REVIEW

1.1 Introduction

The advent of multiprocessor networks has introduced potential computing capabilities previously undreamed of. Before we can make full use of these capabilities, however, we have much to learn about the nature of parallel computing. It is well known that although in the ideal situation system throughput increases linearly with the number of processors, in reality this does not occur. In fact, often after the first few processors, adding additional processors to a system may decrease system throughput. The reason for this, according to Chu et al. [Chu 1980], is interprocessor communication; users tend not to schedule the tasks among processors in such a way as to minimize final completion time as determined by actual task computation time and communication between processors.

In this thesis, I study the difficulty of scheduling multitask jobs on multiprocessor systems to minimize completion time and communication cost. A multitask job is represented by a directed acyclic graph (dag) in which each node represents a task, and an arc from node v to node u means that node v must be computed before node u, and the result of computation of v must be known by the processor computing u. To obtain a schedule that minimizes both processing completion time and communication cost (defined as the number of times that any processor must pass the result of any of its computations to another processor so that the second can compute a task) is an NP-hard problem. I have shown this to be true for arbitrary graphs on two processors, although Coffman and Graham have devised a polynomial time algorithm for the problem when communication is not considered [Coffman 1972]. Furthermore, I show the problem is still NP-hard for graphs with in-degree limited to two. Although the case of two processors computing a graph limited to a general tree remains an open problem, I have upper

and lower bounds on communication for minimum time schedules for complete binary and ternary trees. I determine the lower bound for ternary trees by showing a new partition result for complete ternary trees using a combinatorial method that may be of independent interest.

1.2 Scheduling

Scheduling and task distribution (among processors) are two problems that have been studied fairly extensively in the last ten years, although much of the work on the two problems has been disjoint. Section 1.2 addresses scheduling literature. In Section 1.3, I review some of the research in communication minimization. In studying task scheduling, we view the overall job as a directed acyclic graph, where each node in the graph represents one task, or part of the overall job to be completed. Thus, when I use the term *node*, I will mean the task represented by that node. Arcs in the graph represent precedence constraints, i.e., (u, v) is in the graph if task u must be computed before computation of task v can be started. Thus, the graph specifies a partial order on the tasks. In the general (unit time execution) scheduling problem as defined by Ullman [Ullman 1975] we are given a set S of tasks, a partial order $\sqsubseteq$ on S expressing precedence constraints, a number k of identical processors, and a time limit t. The question is whether there is a total function $f : S \rightarrow \{1, 2, \ldots, t-1\}$ such that

1. If $u \sqsubseteq v$, then $f(u) < f(v)$ $\forall u, v \in S$.
2. $\forall i, 0 \leq i < t$ there are at most k values of v $(v \in S)$ for which $f(v) = i$.

Paraphrased, the problem is to determine whether the tasks can be scheduled on the k processors within time t such that for any two tasks u and v in S, if $u \sqsubseteq v$, then u is scheduled before v, and at each instant, each processor is assigned to at most one task. Ullman shows that this unit time precedence-constrained scheduling problem is NP-complete. Furthermore, he goes on to show that two-processor scheduling with weights of 1 and 2, that is, the problem stated above extended to allow tasks either one or two units of time to compute, is NP-complete.

This last result is moderately surprising since Coffman and Graham construct a polynomial time algorithm for two processor scheduling if all tasks have equal execution times [Coffman 1972]. The algorithm for computing the schedule consists of two parts. First the tasks are all labeled with positive integers in such a way that $\forall u \in S, \forall v \in S$ such that $u \sqsubseteq v$, the label of u is greater than the label of v. The second part of the algorithm is processor assignment. Whenever a processor becomes available, it computes the task u with highest label such that there is no task $v \sqsubseteq u$ that has not yet been computed. Ullman's results show that simply allowing some jobs to require computation time of 2 makes the problem significantly more difficult.

More recently, Gabow has developed an almost linear time algorithm for unit execution time two processor scheduling [Gabow 1982].

If we require the graph representing the partial ordering for the unit time execution scheduling problem to be a tree, we can again design a polynomial time algorithm for the problem regardless of the number k of processors [Hu 1961]. Hu gives an algorithm for scheduling indirected trees, i.e., trees T with root v such that for all nodes u in T, $u \sqsubseteq v$, in linear time. I have not been able to devise a poly-time algorithm for the same problem taking communication costs into account. Like the Coffman-Graham two processor scheduling algorithm, Hu's algorithm requires labeling all nodes of the graph with integers. In this case, each node u is given the label $d(u) + 1$ where $d(u)$ is the length of the path from u to the root. For the actual schedule, define an *available* node to be any node u such that all of u's predecessors have been computed; then at each time unit compute the k available nodes with highest label. In the case of ties, decisions are arbitrary. If fewer than k nodes are available, then compute all of the available nodes.

1.3 Communication Minimization

Whereas scheduling problems concern determining at what time each of several or many tasks should be computed, communication minimization problems concern processor allocation. Much of the previous work in this field does not consider precedence constraints among the various tasks at all, but aims primarily at minimizing computation cost. The system has two kinds of nonnegative contingent costs: a *processing cost* for each pair (P,u) where P is a processor and u a task in the system, and a *communication cost* for each pair (u,v) where u and v are tasks. The processing cost is incurred when task u is computed on processor P, while the communication cost is incurred when tasks u and v are computed on different processors. If the communication cost for a given pair (u,v) is infinite, then nodes u and v must be computed by the same processor. Similarly, if the processing cost for a pair (P,u) is infinite, then node u cannot be processed on processor P. *Total inter-processor communication cost* is the sum of all communication costs incurred in a given schedule. The computation cost is defined as the sum of the processing cost for each task on the processor to which it is assigned, and total inter-processor communication cost.

Chu et al. present a number of strategies for the task allocation problem [Chu 1980]. The first strategy takes a network flow approach to minimizing total cost (as defined above) in a two processor system [Stone 1977]. In this method, the entire job is represented as a graph with each node representing a task to be performed. The graph has additional nodes P and Q, one for each of the processors. Unlike the directed graphs used for the scheduling problem, in this graph, the edges are undirected and weighted. For every pair of tasks u and v, the graph has an edge (u,v) with weight equal to the incurred communication cost when u and v are computed by different processors. Furthermore, for each task u, the graph has edges (P,u) and (Q,u) with weights described below. Edge (P,u) has weight equal to the processing cost (Q,u). Similarly, edge (Q,u) has weight equal to processing cost (P,u). Minimizing total cost as defined above is a

minimum cut problem for this graph.

Unfortunately, this approach is quite limited since it handles only two processor systems. The next approach presented by Chu et al. [Chu 1980], an integer programming approach, is much more flexible and expandable. In this method, communication costs between all pairs of tasks are represented in a volume matrix, and the objective function is the sum of individual processing costs and communication costs. This method is more flexible by virtue of the fact that it allows more than two processors, and other limits can be programmed into the system simply by adding constraint equations for each. Of course, the major disadvantage of this approach is that the general integer programming problem is well known to be NP-hard [Garey 1979].

Lo, in her doctoral thesis [Lo 1983], also addresses the problem of minimizing communication cost in a multiprocessor system. She presents a heuristic algorithm, Algorithm A, for obtaining a near-optimal processor assignment for systems with more than two processors. Algorithm A consists of three parts: Iterative, Lump, and Greedy. Iterative is a generalization of the network flow approach suggested earlier to give an optimal assignment for "most" of the tasks. Lump determines a lower bound on the total cost of a k-way cut (where k is the number of processors) of the remaining unassigned tasks, and based on this number decides either to lump all of the tasks in a group assigned to one processor or to let Greedy complete the assignment. Greedy clusters those tasks between which communication costs are "large" and assigns all tasks in a single cluster to a single processor.

None of the approaches suggested thus far address the question of overall completion time. Processing costs for individual tasks on given processors are treated as constants, and all processors are identical. These algorithms would simply assign all tasks to one processor and save all communication costs. This is not a realistic approach since the goal in minimizing communication cost is ultimately to minimize overall completion time. Lo also addresses this

issue, however [Lo 1983]. She points out that it is very difficult to schedule n tasks on k processors (n and k positive integers) in a balanced way. Her model for scheduling to minimize communication and completion time includes a set S of n (disjoint) tasks each with finite execution time, k identical processors, and contingent communication costs c_{uv} for every pair of tasks (u, v), u and $v \in S$. The cost of a processor allocation A is defined to be the maximum over all processors P of the sum of processing costs of all tasks computed by P and the communication costs incurred by tasks computed by P. The problem, then, is to devise an allocation to minimize this maximum over all possible processor allocations for the given tasks (i.e., it is a minimax problem). Lo points out that even without the contingent communication cost constraint, the problem is NP-hard. The addition of communication costs produces a more difficult problem. If, however, execution times for all jobs are identically equal to a constant t, and similarly if the contingent communication cost between every pair of tasks is equal to a constant c, then if $\frac{t}{c} > \left\lfloor \frac{k}{n} \right\rfloor$, an optimal schedule is determined by assigning the tasks in a round robin manner; if instead $\frac{t}{c} \leq \left\lfloor \frac{k}{n} \right\rfloor$, then an optimal assignment consists of computing all tasks on a single processor. Lo goes on to give some heuristic results for cases when computation times and contingent communication costs are not as well behaved.

1.4 Other Related Communication Results

Papadimitriou and Sipser investigate communication from a slightly different angle [Papadimitriou 1984a]. They investigate a two processor system that uses $2n$ input bits divided between the processors such that each processor has an n-bit input string. They describe a communication complexity hierarchy based on the number of bits that must be passed between the two processors in order to solve a problem (expressed in terms of language recognition) that is a function of all $2n$ bits. COMM($f(n)$) is defined to be the set of languages that can be recognized by passing exactly $f(n)$ bits between the two processors, regardless of the original

partition of the $2n$ bits between them. Papadimitriou and Sipser go on to establish several facts about the communication complexity hierarchy that parallel the classic time-space hierarchy. For example, $COMM(n-1) \neq \emptyset$, and $NCOMM(f(n)) \subset COMM(2^{f(n)})$.

Papadimitriou and Tsitsiklis address the problem of dividing a job's inputs between two processors assigned to compute it in order to minimize communication necessary between the processors in order to complete the job. This is an NP-hard problem [Papadimitriou 1982]. Comparing this result with earlier ones, we see that not only is it difficult to minimize communication necessary between processors when contingent communication between tasks is known, but minimizing those contingent costs is also difficult.

1.5 Other Related Work

Thus far, we have considered time scheduling and processor allocation entirely disjointly from all other issues, although Chu et al. do point out that the linear programming approach to minimizing overall cost considering communication costs is expandable [Chu 1980]. Garey and Johnson consider a similar issue in time scheduling, i.e., the problem of scheduling to minimize completion time when there are additional resource constraints. In this model we are given a set S of tasks and a number k of processors. Each $u \in S$ has an associated completion time $c(u)$, and for each of r resources, a nonnegative resource requirement $R_i(u)$. The tasks are related by a partial order $\sqsubseteq$ as in scheduling problems discussed earlier. Furthermore, we are given a time limit t and an integer b_i for each resource R_i, $1 \leq i \leq r$. The b_i's represent the amount of the associated resource available in the system. The problem is to achieve a schedule defined as a function $f: S \rightarrow \{1, \ldots, t-1\}$ with the following stipulations:

1. $\forall u, v \in S$, $u \sqsubseteq v$ implies $f(u) + c(u) \leq f(v)$.
2. $\forall u \in S$, $f(u) + c(u) \leq t$.
3. $\forall 1 \leq i \leq t-1$, (the number of jobs u such that $f(u) \leq i < f(u) + c(u)$) $\leq k$.
4. $\forall 1 \leq i \leq r$, $\forall 1 \leq j \leq t-1$, (the sum of $R_i(u)$ for u such that $f(u) \leq j < f(u) + c(u)$) $\leq b_i$.

Let U be the set $\{u \in S$ such that $f(u) \leq j < f(u) + c(u)\}$.

Then

$$\forall i, j,\ 1 \leq i \leq t,\ 1 \leq j \leq r,\ \sum_{l} R_j(u) \leq b_j.$$

Since the general scheduling problem is NP-complete, so is this limited resource scheduling problem. Garey and Johnson prove that it is still NP-complete when $k = 2$, $c(u) = c(v)\ \forall$ $u, v \in S$ and $\sqsubseteq$ defines a forest whenever $r > 0$. In contrast, if $k = 2$, c is a constant function, and $\sqsubseteq$ is empty, then $\forall r > 0$ there is a polynomial time algorithm to schedule the tasks. If $k = 3$, then the problem again becomes NP-complete, even for $r = 1$.

1.6 Scheduling and Communication Minimization

The problem of combining finding a minimal time schedule with a minimal communication processor allocation is, of course, at least as difficult as either of its parts, and is a combination of some of these previous problems. Also, to speak of "minimizing time and communication" can be misleading. If all tasks are computed by the same processor, then communication costs are eliminated. Unfortunately, though, the system may not be able to complete the entire job in minimum time with most of the processors idle during the entire computation. Papadimitriou and Ullman discuss this time-communication tradeoff in relation to diamond dags [Papadimitriou 1984b]. A diamond dag is a directed acyclic graph (dag) that is the generalization of Figure 1 below to n^2 nodes. Each node of the dag represents one task needing computation, and the edges specify a partial order on the tasks. Additionally, the communication cost of a schedule and processor allocation is defined to be the number of processor, node, immediate predecessor triples (P, n, p) such that processor P first computes node n before it has computed node p (although p must have been computed at some earlier point by a different processor) or if it never computes p. Papadimitriou and Ullman allow nodes to be computed more than once in this model. Relating this to other models studying communication, we can imagine that an arc from node u to node v represents an contingent

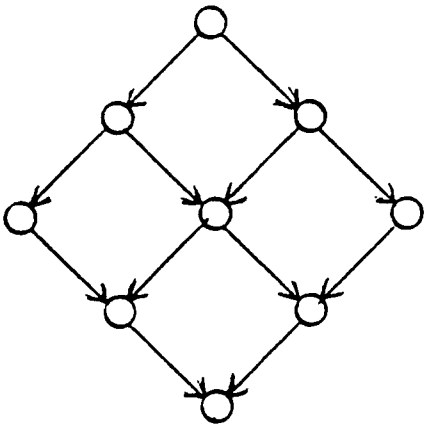


Figure 1:
3 × 3 Diamond dag

communication cost of 1, whereas the lack of such an arc represents an contingent communication cost of 0. If each node is computed only once, then the analogy is completely parallel: cost c_{uv} is incurred whenever nodes u and v are computed by different processors, that is, when node v is computed by a processor that does not compute node u. Since multiple computation of nodes is allowed, c_{uv} is incurred whenever the *first* computation of node v is by a processor that has not (yet) computed node u. Using this model, Papadimitriou and Ullman derive time-communication tradeoffs for this specific kind of graph (or partial order). In particular, any schedule computing an $n \times n$ diamond dag in time $t = o(n^2)$ and communication c must satisfy $ct = \Omega(n^3)$

Aggarwal and Chandra also study communication-time tradeoffs for scheduling dags [Aggarwal 1985]. Their model uses a concurrent read exclusive write PRAM in which each processor has a local memory and all processors share a global memory. The computation is divided into computation and communication steps. In a computation step, each processor can access two local memory addresses, whereas in a communication step, each processor can access one global memory address. All inputs are in the global memory, and all final outpouts must be returned to the global memory. Aggarwal and Chandra use this model to study minimum

communication delays, i.e., the minimum number of communication steps necessary, for particular computations; for instance, when two $n \times n$ matrices are multiplied using only scalar multiplications and additions, there is a bound on communication delay of $\theta\left\lfloor \frac{n^2}{k^{\frac{2}{3}}} \right\rfloor$ where k is the number of processors, and $k \leq \frac{n^3}{\log^2 n}$. Aggarwal and Chandra also study communication-time tradeoffs. They note that the tradeoff for diamond dags presented by Papadimitriou and Ullman [Papadimitriou 1984b] is obtainable only for $t = 1$ or $t = n$.

From this point forward I will use the term *schedule* to refer to the time schedule and the processor allocation simultaneously. Afrati et al. address the problem of finding a minimum time and communication schedule for any given graph [Afrati 1985]. They use a model similar to that of Papadimitriou and Ullman except that they do not limit the graph to diamond dags and they allow each node to be computed only once. Thus, for a given graph $G = (V, A)$ and schedule D, the communication cost is defined to be the number of pairs of nodes (u, v) such that $u, v \in V$, $(u, v) \in A$, and u and v are computed by different processors in D. This parallels Papadimitriou and Ullman's definition of communication when multiple computation of nodes is not allowed. Since scheduling in general is NP-complete whenever $k > 3$ and the partial order on the nodes is arbitrary, we cannot expect better for minimum time and communication scheduling. Afrati et al. show that with two processors on a general graph, the problem is still NP-complete. Furthermore, if the graph is a tree and the number of processors is unlimited, the problem is also NP-complete.

Chapter 2

DEFINITIONS AND NOTATIONS

2.1 The Model

For the balance of this work, I consider a multiple task job to be modeled as a directed acyclic graph, or *dag*. In this graph, each node represents a task, or part of the overall job to be computed, and all tasks require equal computation time. Arcs represent precedence constraints. If an arc (u, v) appears in the graph, then computation of node v depends on the result of computation of node u; i.e., node u must be computed before node v, and the result of computation of node u must be known by the processor computing node v. This appears to be a reasonable form of representation, particularly in regard to a straight-line program, as pointed out by Tompa [Tompa 1980]. Computation of the graph is by a number of identical processors. Now I introduce several definitions that will be used throughout the rest of the work to facilitate discussion of scheduling such graphs on identical processors and the communication cost incurred in so doing.

2.2 General Graphs and Multiprocessor Schedules

The first few definitions regard general directed acyclic graphs and schedules for processing them. We let $G = (V, A)$ be a dag with schedule and processor allocation S on k processors.

processor schedule: A *processor schedule* on k processors for G is a function

$f : V \rightarrow \{1, 2, \ldots, k\} \times N$ (N is the set of natural numbers) such that

i) there is no $u \neq v \in V$ such that $f(u) = f(v)$, and

ii) for (u, v) such that $(u, v) \in A$, if $f(u) = (i, j)$ $f(v) = (m, n)$ then $j < n$

processor assigned to v: If $f(v) = (i, j)$ for some $v \in V$ then we say that processor i is the *processor assigned to* v, and v is scheduled to be processed at time j.

proc(v): For every $v \in V$, *proc(v)* denotes the processor assigned to v in S.

n(P): For each processor P, $n(P)$ denotes the number of nodes in V assigned to P in S.

communication node: A node $v \in V$ is said to be a *communication node* if there is a node $w \in V$ such that $(v,w) \in A$ and $proc(v) \neq proc(w)$.

communication receiver: A node $v \in V$ is said to be a *communication receiver* if there is a node $w \in V$ such that $(w,v) \in A$ and $proc(v) \neq proc(w)$.

communication to v: For every $v \in V$, the *communication to v* is the number of nodes w such that $(w,v) \in A$ and $proc(v) \neq proc(w)$.

available node: At any time t during the computation of a graph, an *available node* is defined to be any node v such that all nodes w such that $(v,w) \in A$ have already been computed; that is, if $f(w) = (i,j)$ then $j < t$.

2.3 Communication Cost

I present two definitions of communication cost.

Definition 1: For a dag $G = (V,A)$ and processor schedule S, *communication cost* is defined to be the number of pairs u,v such that $(u,v) \in A$ and $proc(u) \neq proc(v)$.

Definition 2: For a dag $G = (V,A)$ and processor schedule S, *communication cost* is defined to be the number of processor node pairs (P,v) such that processor P does not compute node v but computes at least one direct successor of v.

Papadimitriou and Ullman [Papadimitriou 1984b] and Afrati et al. [Afrati 1985] both use Definition 1 of communication in their work. I introduce Definition 2 as a more practical definition of communication cost. By counting each arc (u,v) such that $proc(u) \neq proc(v)$ as

communication using Definition 1 we may be passing the same information between a given pair of processors several times. For instance, if nodes u, v, and $w \in V$ such that $(u,v), (u,w) \in A$ and $proc(v) = proc(w) \neq proc(u)$, then the result of computing u must be communicated to the processor computing v and w. This communication is counted in both definitions, however Definition 1 of communication counts it twice. Definition 2 counts it only once. Definition 2 counts only the number of results that must be passed from one processor to another, and assumes that the second processor can save each value for use in processing all nodes requiring it.

2.4 Trees

par(v): For any node v, *par(v)* will denote the parent of v. In an indirected tree T, par(v) is the node w such that w is directly above v in T, that is the arc (v,w) is in T.

child(v): For a node v, *child(v)* denotes the child of v, that is, the node w such that v = par(w).

rc(v): In an ordered tree, *rc(v)* denotes the rightmost child of the node v.

lc(v): In an ordered tree, *lc(v)* denotes the leftmost child of the node v.

CHAPTER 3

MINIMUM TIME AND COMMUNICATION SCHEDULING ON TWO PROCESSORS

3.1 General Graphs on Two Processors

Although general minimum time scheduling on two processors can be done in polynomial time [Coffman 1972], [Gabow 1982], when a communication bound is introduced the problem again becomes more difficult. Afrati et al. [Afrati 1985] showed that with Definition 1 of communication, this problem is NP-complete. Now I turn to Definition 2 of communication. Whereas with Definition 1 we count communication arcs, in the two processor case using Definition 2 we can count communication nodes. Though in general using Definition 2 of communication a node could introduce more than one unit of communication if more than one other processor requires information from computation of that node, in the two processor case, only one processor other than the one computing any node might need the result. Ergo, we have the following lemmas concerning two processor schedules on a graph $G = (V, A)$.

Lemma 3.1

If there are nodes u and v such that $proc(u) \neq proc(v)$, then any node x such that x is an immediate predecessor of both u and v is a communication node.

Proof

If x is an immediate predecessor of u and v, then by definition, $(x, u) \in A$, and $(x, v) \in A$. If $proc(x) = proc(u)$ then $proc(x) \neq proc(v)$. Similarly, if $proc(x) = proc(v)$, $proc(x) \neq proc(u)$. In any event, x is a communication node.

□

Lemma 3.2

If there are nodes u and v such that proc(u) $\neq$ proc(v) and there is a node x such that x is an immediate successor of both u and v, then either u or v is a communication node.

Proof

If proc(x) = proc(u), then proc(x) $\neq$ proc(v). Consequently, v is a communication node. Similarly, if proc(x) = proc(v), then u is a communication node.

□

Lemma 3.3

If there are nodes u and x such that x is a successor of u (not necessarily immediate), and proc(u) $\neq$ proc(x), then some node in the path from u to x is a communication node.

Proof

I use induction on the length of the path from u to x. Suppose x is an immediate successor of u. Then by definition of communication, u is a communication node. Now suppose that Lemma 3.3 is true for any path of length $k - 1$, and the path from u to x has length k. If for some node v such that v is a child of u, v is on the path from u to x, and proc(v) $\neq$ proc(u), then u is a communication node. If proc(v) = proc(u), then proc(v) $\neq$ proc(x), and the length of the path from v to x is $k - 1$. Then by the inductive hypothesis, some node on that path is a communication node.

□

Corollary 3.1

If nodes u and v such that proc(u) $\neq$ proc(v) have a common successor x, then at least one node in the path from u to x or that from v to x is a communication node.

Proof

This follows directly from Lemma 3.3. If proc(u) = proc(x), then a node on the path from v to x is a communication node. Similarly, if proc(v) = proc(x), then a node on the path from u to x is a communication node.

□

Corollary 3.2

If nodes u and v such that proc(u) ≠ proc(v) have a common predecessor x, then at least one node in the path from x to u or in that from x to v is a communication node.

Proof

This, too, follows directly from Lemma 3.3. If proc(x) = proc(u), then a node on the path form x to v is a communication node, otherwise a node on the path from x to u is a communication node.

□

Corollary 3.3

Suppose there are nodes, u, v, w, and x such that v and w are in disjoint paths from u to x. If v and w are computed by different processors, then to compute the subgraph defined by all paths from u to x requires a minimum communication cost of 2.

Proof

By Corollary 3.1 either the path from v to x or the path from w to x has at least one communication node, and by Corollary 3.2 either the path from u to v or that from u to w has a communication node. Then the whole structure has at least two communication nodes, hence communication cost is at least 2.

□

Now I am ready to introduce the problem. From this point on, whenever I speak of communication I am referring to Definition 2 of communication.

Two Processor Scheduling(2PS)

INSTANCE: Given a dag $G = (V, A)$ and integers c and t.

QUESTION: Can G be scheduled on two processors within time t and communication c?

Theorem 3.1

2PS is NP-complete.

Proof

To show that 2PS is NP-complete I reduce 3SAT to it. Recall that in an instance (X, S) of 3SAT we are given a set $X = \{x_1, x_2, \cdots, x_n\}$ of variables, and a set $S = \{s_1, s_2, \ldots, s_m\}$ of clauses, where each clause is of the form $s_j = (z_{j1}, z_{j2}, z_{j3})$ such that $z_{ji} = x$ or $\bar{x}$ for some $x \in X$ and $\forall i, j$, $1 \leqslant i \leqslant 3$, $1 \leqslant j \leqslant m$. Furthermore, for each j, the literals z_{j1}, z_{j2}, and z_{j3} are distinct, and for no $x \in X$ are both x and $\bar{x}$ in the same clause. The question is whether the variables have a truth assignment causing at least one literal in each clause to be true.

First, for each $x_i \in X$ construct a four node *gadget* as in Figure 2. The four nodes for each variable are a_i, x_i, $\bar{x}_i$ and b_i. Node a_i is the source node, with arcs to the two center nodes x_i and $\bar{x}_i$ representing the variable and its complement which in turn have arcs to node b_i, the sink node. Next, for each $s_j \in S$ construct a five node *widgit* as in Figure 3. Again, there is a source node a_{n+j} with arcs to the three center nodes z_{j1}, z_{j2} and z_{j3}. The three center nodes of each clause widgit are associated with the three literals in the clause. They in turn have arcs to the sink b_{n+j}. Note that the numbering of the a and b nodes of the widgits starts at $n+1$ [illegible] so a consecutive numbering of all gadgets and widgits is maintained.

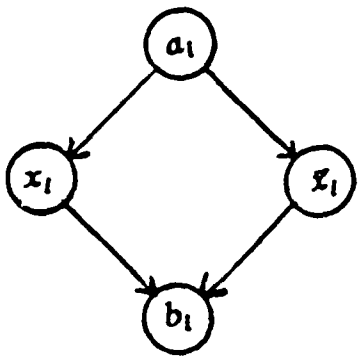


Figure 2:
"Gadget" for node x_l

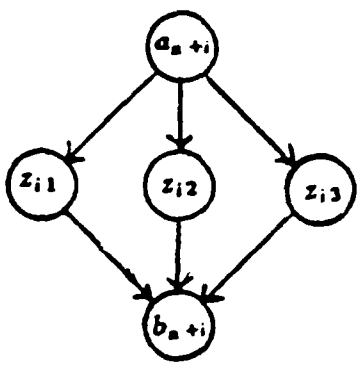


Figure 3:
"Widgit" for clause s_i

To construct the graph for 2PS, connect all of the gadgets followed by all of the widgits into one column, identifying a_i with $b_{i-1}, \forall i$, $1 < i \leq n + m$. Consequently, the source of one gadget or widgit is identified with the sink of the previous one in the column. Add

additional arcs from the variable gadgets to the clause widgits as follows. Let $s_i \in S$ be (z_1, z_2, z_3). If $z_j = x_k$ for some $1 \leq j \leq 3$, $1 \leq k \leq n$, then introduce an arc from node x_k to node z_j. Similarly, if $z_j = \bar{x}_k$ for some $1 \leq j \leq 3$, $1 \leq k \leq n$, then introduce arc $(\bar{x}_k, z_j)$. Last, add the apex AX and arcs (AX, a_i) for all $1 \leq i \leq n + m$ as well as an arc (AX, b_{n+m}). That is, AX is a single node with arcs going to all of the source/sink nodes in the column. This completes the graph for our instance of 2PS. Now set

$t = 2n + 3m + 2$, $c = 2n + 2m$.

As an example, consider the following instance of 3SAT:

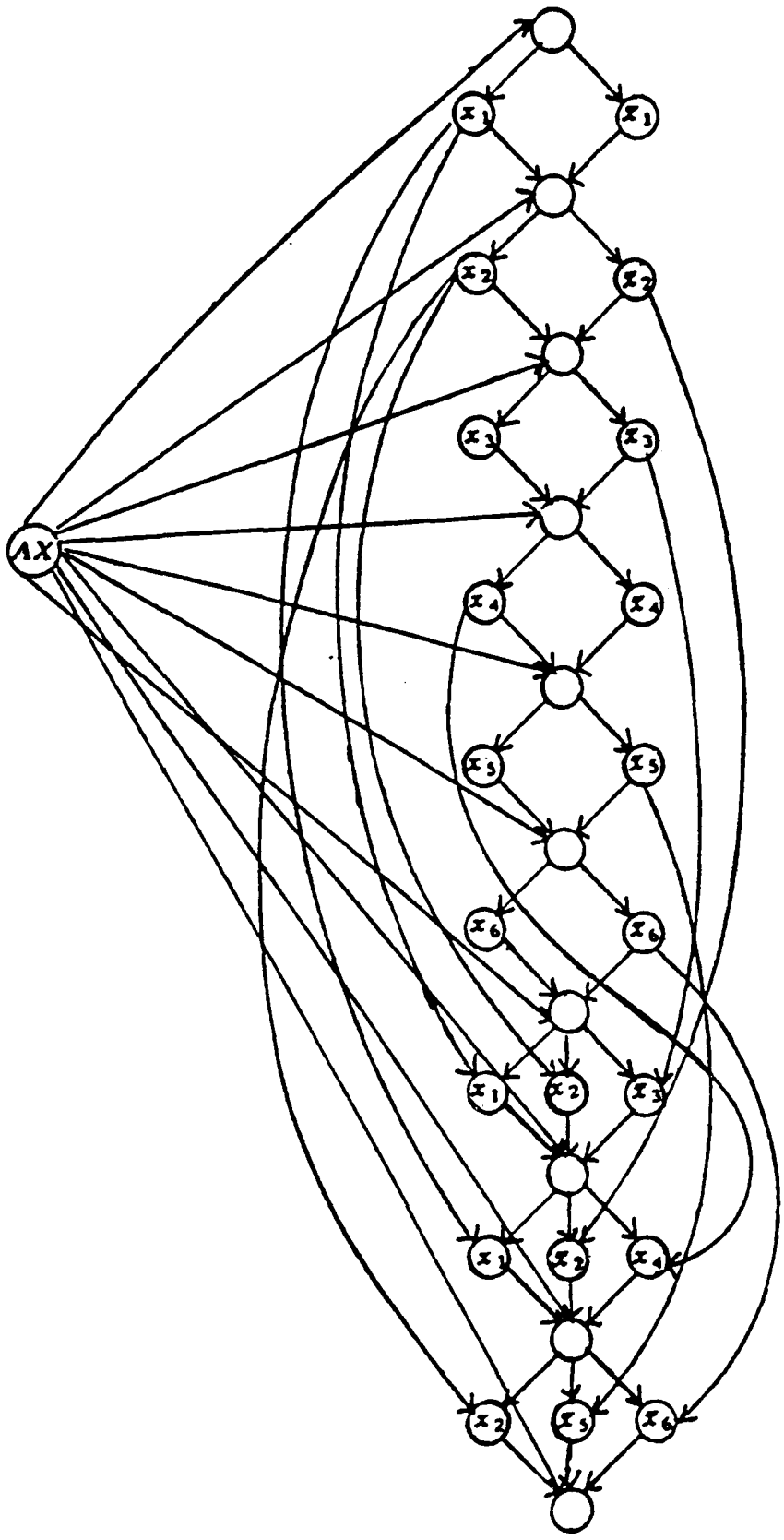
3SAT example

$X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$

$S = \{(x_1, x_2, \bar{x}_3), (x_3, \bar{x}_2, x_4), (x_2, \bar{x}_5, \bar{x}_6)\}$

Figure 4 illustrates the proposed graph representing this instance of 3SAT.

To show that an instance of 2PS constructed in this way is equivalent to the original 3SAT instance, suppose that the instance of 3SAT has a satisfying truth assignment T. First, assign the apex and all of the a and b nodes to the first processor (P). For each variable x_i, if x_i is assigned the value *false* in T then assign x_i to processor P for computation; otherwise, assign $\bar{x}_i$ to P. Assign all of the other nodes in the gadgets to the second processor (Q). In addition, assign to Q one of the center nodes in each clause widgit associated with a *true* literal. If more than one are *true* then choose one arbitrarily. Assign all other widgit centers to processor P. Thus, processor Q is assigned exactly one center node of each variable gadget and each clause widgit and nothing else, and the value in T assigned to each of these is *true*. This processor allocation can be executed within the required time limit in the following way. Processor P computes AX at time 1, and Q computes its assigned nodes until the entire graph is computed. Then P computes a_1 at time 2, and a_1 becomes available. So on, for each $1 \leq i \leq n$, if the variable gadget x_i is assigned to P then at time $i + 1$ the first time step processor P computes a_i at



$t = 23$; $c = 18$

Figure 4:
Graph for 3SAT example

time $2i$. Each processor computes one gadget center at time $2i + 1$. For all j, $1 \leq j \leq m$ compute the widgit associated with clause c_j by the schedule: P computes $a_{j,*}$ at time $2n + (3j - 1)$. The three center nodes are computed by the two processors at times $2n + 3j$ and $2n + (3j + 1)$. The order of their computation is arbitrary. The final node $b_{*,*}$ is computed by processor P at time $2n + 3m + 2$ which is the required limit.

To determine the communication cost, count the communication nodes. Since both processors work on all centers, by Lemma 3.1 each source is a communication node. (The apex is not a communication node.) Also, each center node computed by Q is a communication node since it has an arc to a sink that is computed by P. The only other nodes in the graph are the center nodes computed by P. I claim that none of these are communication nodes. Each of these nodes has an arc to the sink of a gadget, also computed by processor P, and perhaps has arcs to center widgit nodes. The processor assignment was such that Q only computes widgit centers that have the value *true* in T and therefore have processor Q computing their corresponding gadget centers. Then no gadget center computed by P has an arc to any widgit center computed by Q. The widgit centers have arcs only to source sink nodes. Since P computes all of the a and b nodes, no widgit center computed by P is a communication node; ergo no center node computed by P is a communication node.

Counting the communication nodes, we find $n + m$ for the sources and $n + m$ for the centers computed by Q. This totals $2n + 2m$, which is our communication bound.

Now suppose the instance of 2PS constructed above has a solution. To show that the corresponding instance of 3SAT is satisfiable I introduce another lemma.

Lemma 3.4

At any time in the computation of any schedule, either

- The apex is computed, or

ii) A single source or sink node is computed, or

iii) One or two center nodes are computed; if two, both are the center of the same gadget or widgit.

Proof

Since the apex is the only node of in-degree 0, only AX is computed at time 1, therefore (i) is trivial. Since the source/sink nodes are ordered, $\forall i, j$, if $i < j$, then a_i is a predecessor of a_j, so a_i must be computed before a_j. Node $b_{i,+,n}$ is a successor of all of the a nodes and, therefore, must be computed only after the rest have been finished. Therefore, no two source sink nodes can be computed at the same time. Similarly, each source sink node is either a predecessor or successor of each of the center nodes and therefore no source/sink can be computed simultaneously with a center node. Thus, no other node can be computed at the same time as any single source/sink node.

Since the center nodes cannot be computed at the same time as any other kind of node (by (i) and (ii) above) if one processor is computing a center node the other must either be idle or also be computing a center node. If two center nodes are computed concurrently, they must both be from the same gadget or widgit since for every gadget or widgit y, the center nodes of every other gadget or widgit in the column are either predecessors or successors of all nodes in y.

□

Corollary 3.4

The minimum time for computing the center of a gadget is 1, the minimum for a widgit center is 2, and to obtain these minima both processors must work.

Proof

Since each gadget has two center nodes, the only way to compute both in time 1 is for each of the two processors to compute one of them. Similarly, since widgits have three center nodes, one of the two processors must compute two of them causing the minimum time to be two. If one processor did all three the time would be three.

□

By Lemmma 3.4 and Corollary 3.4, the minimum time for computing the graph is

1 .	(apex)
$+ n + m + 1$	(source/sinks)
$+ n$	(gadget centers)
$+ 2m$	. (widgit centers)
$= 2n + 3m + 2.$	

This is the time bound given in the instance of the problem. By Corollary 3.4 we see that to obtain the minimum time for each center, both processors must work. Thus, in any solution to this instance of 2PS, both processors are active on every center. Then by Corollary 3.3 the communication within each gadget and widgit is 2, hence the total communication in the gadgets and widgits is $2(n + m)$. This, too, is the bound expressed in our instance of 2PS. Then the apex cannot be a communication node since all communication is used in the rest of the graph.

Call the processor computing the apex P. Since the apex is not a communication node, all sources and sinks are also computed by P since for all $1 \leq i \leq n + m$ the graph has arcs (AX, a_i) as well as arcs (AX, b_{n+i}). Since both processors are active on every center, all source nodes are communication nodes. Also, each center node processed by Q is a communication node since it has an arc to a sink computed by P. The total of the source nodes and the minimum number of center nodes is $2(n + m)$, since we cannot afford any more communications than this

none of the centers computed by P is a communication node.

Now make the truth assignment as follows. We know that processor Q computes either x_j or $\bar{x}_j$ but not both $\forall\, j,\ 1 \leq j \leq n$. If Q computes x_j, then assign the value *true* to variable x_j; if Q computes node $\bar{x}_j$, then assign *false* to variable x_j. Since gadget centers computed by P are not communication nodes, Q must compute the gadget center predecessor of every widgit center computed by Q. Consequently, with our truth assignment, each clause has a *true* literal corresponding to the widgit center computed by Q. Since Q computes at least one center node in each clause widgit, each must have at least one *true* literal, and the truth assignment satisfies (X, S).

□

3.2 Limited In-degree Graphs on Two Processors

If we assume that our graph represents a problem with fine grain parallelism, i.e., each node represents an elementary operation such as addition, it makes sense to consider graphs with in-degree limited to two since most elementary operators are either unary or binary. Using the elements of the previous construction with one change, I show that scheduling a graph of in-degree 2 in minimum time and communication is also an NP-complete problem. To do so, I introduce a new subgraph structure, the 11-node modified widgit, or *modgit* which is pictured in figure 5. Label the nodes in the modgit A through K, starting with the source and labeling from left to right on each level (in Figure 5). Nodes B, C, and D all have arcs coming to them from node A. Now let nodes B through J be the center nodes with B, C, and D being the *crucial centers*. Node A has in-degree 0, the crucial centers and nodes I and J have in-degree 1 whereas all other nodes in a modgit have in-degree 2.

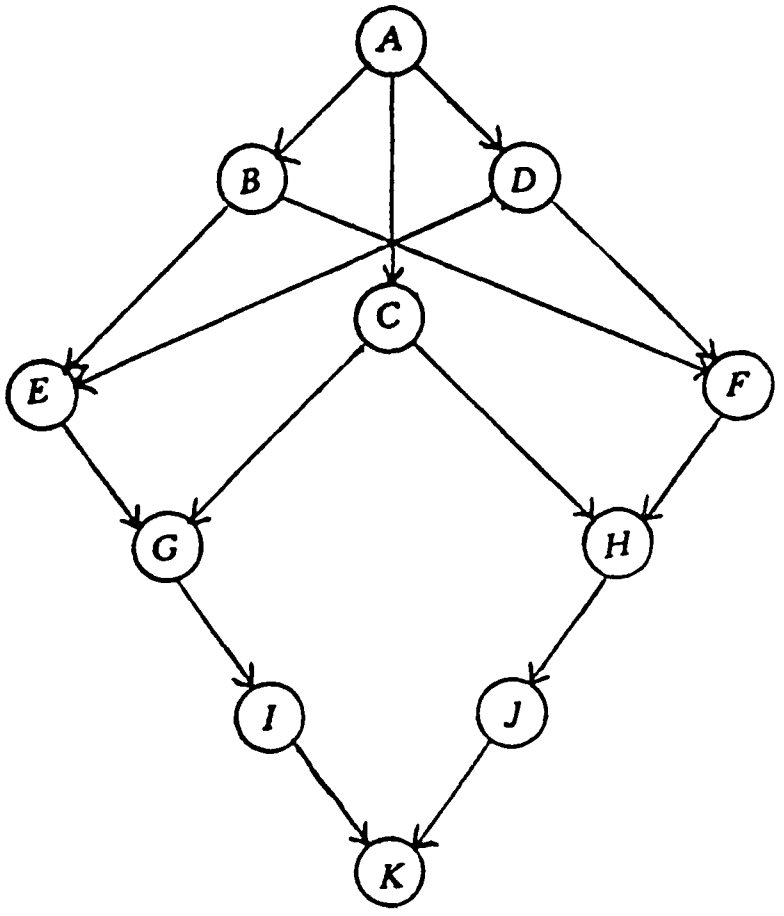


Figure 5:
An 11 node Modgit

Lemma 3.5

A modgit can be scheduled in time 7 on two processors. Furthermore, it cannot be scheduled in time less than 7.

Proof

Since node A must be computed before any other node, inasmuch as it is the only node with in-degree 0 and is therefore a predecessor of the others, and node K cannot be computed until after all other nodes, both nodes A and K must be computed alone, that is one processor

must be idle during computation of each of them. The other nine nodes require time at least 5 to be computed on two processors, so the whole modgit requires time 7. Figure 6 gives an example of a schedule completing in time 7 to complete the proof.

□

Lemma 3.6

To compute a modgit in minimum time on two processors, each processor must compute at least one of the crucial center nodes.

Proof

To maintain minimum time, a processor can be idle for only one time unit other than times 1 and 7. After node A is computed, nodes B, C and D are the only available nodes. Furthermore, no other node becomes available until both B and D have been computed. If all three of the crucial centers are computed by the same processor, then the second processor is idle during computation of both B and D, which renders minimum time unattainable.

□

Time	P	Q
1	A	idle
2	B	C
3	D	idle
4	F	E
5	H	G
6	J	I
7	K	idle

Figure 6:
A processor schedule for a Modgit

Lemma 3.7

Computation of a modgit in minimum time on two processors requires communication cost at least 5.

Proof

I will show this by counting communication nodes. By Lemma 3.6, at least one of the crucial centers is computed by each processor so one processor computes one crucial center and the other computes two. Then by Lemma 3.1, node A is a communication node. We now have two cases.

Case 1: Both I and D are computed by the same processor.

Without loss of generality, let P compute B and D. Node A must be computed at time 1. Both nodes B and D must be computed before either E or F can be computed. Therefore neither E nor F can be computed before time 4 since P must use times 2 and 3 to compute nodes B and D. For the modgit to be computed in time 7, nodes H and G must then be computed at time 5, I and J at time 6, and node A at time 7. Then one node of each pair $\{E, I\}$, $\{G, H\}$ and $\{I, J\}$ must be computed by P and the other by Q, which results in all of the crucial center nodes being communication nodes by Lemma 3.1 since each one has two immediate successors computed on different processors. Similarly, by Lemma 3.2, either I or J must be a communication node since A is a direct successor of both of them. Then there are at least 5 communication nodes: A, E (or D), and either I or J.

Case 2: B and D are computed on the other processor.

Without loss of generality, let P compute B. By Lemma 3.2, at least one of B and D must be a communication node since A is a direct successor of both of them.

Case 2.1: Node B is computed by P and node D by the other processor.

By Lemma 3.1 node C is a communication node, and by Corollary 3.3, at least one of nodes G, H, I, and J must be a communication node.

Case 2.1.1: proc(E) = proc(F).

If proc(E) = proc(G), then proc(F) ≠ proc(H) and vice versa. In either case one of nodes E and F is a communication node. Then the communication nodes are A, C, and at least one each of the sets $\{B, D\}$, $\{E, F\}$, and $\{G, H, I, J\}$.

Case 2.1.2: Nodes E and F are computed by different processors.

By Lemma 3.1, with $u = E$, $v = F$, and $x = B$ and D, both B and D are communication nodes. Then the communication nodes are A, B, C, D, and one of the set $\{G, H, I, J\}$.

Case 2.2: Nodes G and H are computed by Processor P.

Processor Q must compute at least four of the nodes B through J. Consequently, Q must compute at least three of the nodes E, F, I and J, since it computes only one of the other nodes (B). In particular, Q must compute at least one immediate predecessor of G or H, and at least one of their immediate successors. Then the communication nodes are A, and at least one of each of the pairs (B, C), (E, F), (G, H), and (I, J).

Case 2.3: Both nodes G and H are computed by Processor Q.

By the definition of communication, C is a communication node.

Case 2.3.1: Nodes E and F are computed by the same processor.

First note that if proc(E) = proc(F), then proc(E) = proc(F) = Q because otherwise we cannot achieve the time bound. Thus, nodes B, E, F, G, and H are all computed by Processor Q. Then Processor P must compute both I and J. Then communication nodes are A, C, D, G, and H.

Case 2.3.2: Nodes E and F are computed by different processors.

Again by Lemma 3.1, both B and D are communication nodes. Then the communication nodes are A, B, C, D, and either E or F.

In any event, communication of at least 5 is incurred and thus the Lemma is proved.

□

We are now ready for the statement of the problem.

Two Processor Scheduling of Graphs with In-degree Limited to 2 (2PS-2)

INSTANCE: Given a directed acyclic graph $G = (V, A)$ with maximum in-degree 2 and integers t and c.

QUESTION: Can G be scheduled on two processors in time no more than t with communication cost no more than c?

Theorem 3.2

2PS-2 is NP-complete.

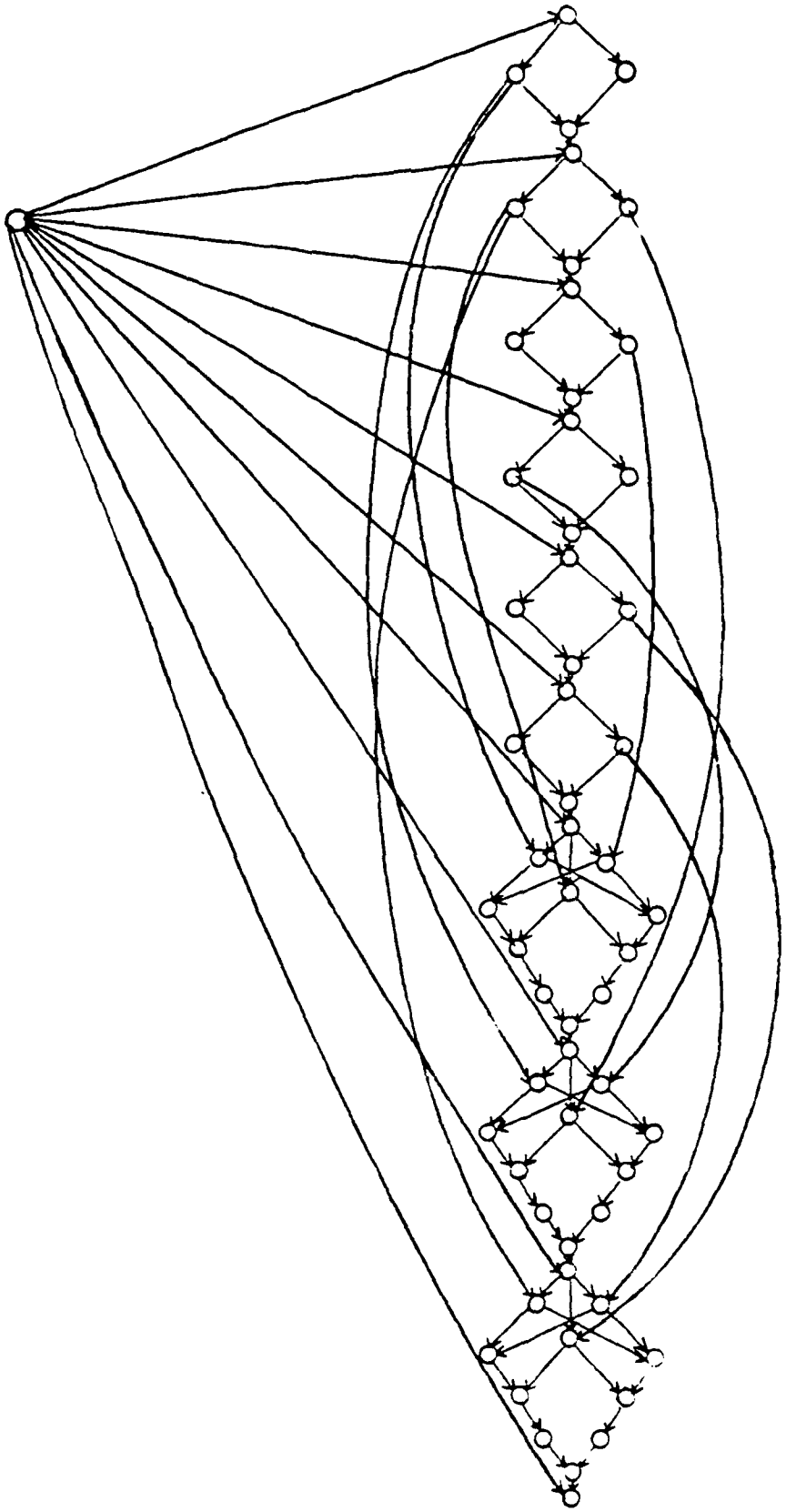
Proof

I will show this using a proof paralleling that used for 2PS. Recall that I reduced 3-SAT to that problem.

To reduce 3-SAT to 2PS-2, again construct a 4-node gadget for each of the n variables (as before). For each of the m clauses, construct a modgit. To make the entire graph, again chain all of the gadgets and modgits in a column; however, in this case do not equate the sink of one with the source of the next. That is, for each $1 \leq i$ [illegible] add an arc from [illegible] to [illegible]. Furthermore add arcs from [illegible] to A_{i+1} [illegible] each $1 \leq$ [illegible] m [illegible] the clause modgits, where L is [illegible] node [illegible] the [illegible] clause [illegible] L in the set [illegible] A [illegible] We [illegible] need to connect the [illegible] the gadgets by [illegible] arc from [illegible] Note that [illegible] all nodes have in-degree no

more than 2, and the sources of all gadgets and modgits have in-degree no more than 1, as do the crucial center nodes of the modgits. Now add one new node, the subsink SS, and the arc (X, SS). Again, add the apex AX, with arcs from AX to all of the source nodes (increasing their in-degree to no more than 2) and arcs to SS, giving it in-degree of 2. In each modgit, associate one of the crucial centers with each literal in the clause, that is B_i represents $z_{i,1}$, C_i represents $z_{i,2}$, and D_i represents $z_{i,3}$ for all i, $1 \leq i \leq m$. Next, as in the case with 2PS, connect the gadgets to the crucial center nodes of the modgits. If $z_{i,k} = x_j$ for some $1 \leq i \leq m$, $1 \leq k \leq 3$, $1 \leq j \leq n$, then add arc $(x_j, z_{i,k})$. Similarly, if $z_{i,k} = \bar{x}_j$ for some $1 \leq i \leq m$, $1 \leq k \leq 3$, $1 \leq j \leq n$, add arc $(\bar{x}_j, z_{i,k})$ where, in both cases, $z_{i,k}$ means the node representing $z_{i,k}$ in the modgit. Now each crucial center node has one additional arc going to it, bringing the in-degree to 2. Let $t = 3n + 7m + 2$, $c = 2n + 5m$. As an example, recall 3SAT example. For the limited in-degree case, the graph looks something like Figure 7. Note again that this is the minimum possible value for t since each gadget must be totally computed in time no less than 3 before the next can be started, and similarly with the modgits, in time no less than 7. The apex and subsink account for the other two time units. Similarly, c is the minimum possible communication within that time since each gadget requires communication 2 to achieve minimum time, and by Lemma 3.7 each modgit requires communication 5.

Now suppose there is a satisfying assignment for the 3-SAT problem. Then the graph can be scheduled within the time and communication bounds as follows. As in 2PS, let P compute the apex and all source and sink nodes. Furthermore, if a variable x is assigned true in the satisfying assignment, then P computes node $\bar{x}$ while Q computes node x. Otherwise, P computes node x and Q computes node $\bar{x}$. Again, pick one true literal from each clause. If the literal picked for a particular clause is $z_{i,2}$, i.e., it corresponds to node C_i, schedule the clause as in Figure 8. Communication within the modgit is 5 with communication nodes A, B, C, D, and J. If, on the other hand, the chosen node is B or D, schedule the modgit as in Figure 9. Again, communication within the modgit is 5 with communication nodes A, B, C, D, and J.



$n = 41$; $\alpha = 2^{-7}$

Figure 7:
Graph for 3SAT example, limited in-degree

time	P	Q
1	A	idle
2	B	C
3	D	idle
4	F	E
5	H	G
6	J	I
7	K	idle

Figure 8:
Schedule if C is "chosen" for Processor Q

time	P	Q
1	A	idle
2	B	D (reversed if B is the chosen node)
3	C	idle
4	E	F
5	G	H
6	I	J
7	K	idle

Figure 9:
Schedule if B or D is "chosen" for Processor Q

Because all center gadget nodes computed by P go to crucial center modgit nodes computed by P, no gadget center computed by P is a communication node, although all those computed by Q are. As in Theorem 3.1, the graph is scheduled by first computing the apex and then keeping P constantly active while Q works one center gadget node and 4 center modgit nodes for each variable and clause respectively.

Conversely, suppose that the graph can be scheduled within time $3n + 7m + 2$ and communication $2n + 5m$. I show that the corresponding instance of 3SAT is satisfiable. As I pointed out, for this graph this is the minimum possible time and the minimum communication for that time. Then each gadget incurs communication 2, each modgit incurs communication 5, and neither the apex nor any of the sink nodes can be communication nodes. Then AX, all sources, all sinks, and SS must be computed by the same processor; without loss of generality,

let it be P. Again, the gadget centers computed by P cannot be communication nodes. Consequently, P must compute every crucial center modgit node whose immediate predecessor gadget center is computed by P. By Lemma 3.6, however, at least one crucial center of each modgit must be computed by Q. Since no center gadget node computed by P is a communication node, the (gadget) immediate predecessor of the crucial center computed by Q must also be computed by Q (see the end of the previous proof). Again make the following truth assignment. If Processor Q computes node x for the variable x, assign *true* to variable x; otherwise assign the value *false* to x. As before, this completes the proof, since the literal associated with the modgit crucial center computed by Q in each case must have the value *true*.

□

CHAPTER 4

SCHEDULING COMPLETE BINARY TREES ON TWO PROCESSORS

Hu [1961] designed a polynomial time algorithm for scheduling computation trees in minimum time regardless of the number of processors, whereas Afrati et al. [Afrati 1985] showed that to minimize communication within that time frame is an NP-complete problem. Moreover, I have shown that scheduling a general graph on two processors in minimum time and communication (within the time frame) is an NP-complete problem. How difficult is it, then, to schedule a tree on two processors in minimum time with minimum communication cost?

I will address only indirected trees. For these trees, since each node except the root has outdegree 1, Definitions 1 and 2 of communication cost are equivalent. In an effort to gain some insight into the problem of scheduling computation trees on two processors in minimum time and communication we study special kinds of trees, specifically complete binary and ternary trees. Chapter 4 investigates complete binary trees, whereas Chapter 5 examines complete ternary trees.

Complete binary trees are, in fact, quite easy to schedule in minimum time and communication on two processors.

Theorem 4.1

The complete binary tree B_k of height k can be scheduled on two processors in time 2^k with communication cost of 1. Furthermore, computation of B_k requires at least time 2^k and communication cost 1 to achieve this time

Proof

First consider optimality. Observe that B_k has $2^{k+1} - 1$ nodes. The time to compute it on two processors, then, must be at least $\left\lceil \frac{2^{k+1} - 1}{2} \right\rceil = 2^k$

To determine the minimum communication, suppose processor Q computes the root x, and processor P computes some other node $v \in B_k$. This must be true for some node v since to optimize time each processor must compute half of the non-root nodes. By Lemma 3.3, at least one node on the path from v to x is a communication node and communication is thus at least 1.

The schedule for computing B_k in optimal time and communication is simple. For each node v, let $proc(v) = P$ if v is in the right subtree of the root, $proc(v) = Q$ otherwise. Processor Q computes the root. This guarantees communication of 1 since rc(root) is the only communication node. Each processor computes the nodes in its subtree one at a time from the leftmost lowest node to the root of the subtree. Since each subtree has $2^k - 1$ nodes, this part of the computation requires time $2^k - 1$. At time 2^k, Q computes the root.

□

CHAPTER 5

SCHEDULING COMPLETE TERNARY TREES ON TWO PROCESSORS

5.1 An Upper Bound for Com .unication Cost (Schedule)

Complete ternary trees offer more of a challenge and consequently more insight into the overall difficulty of scheduling general trees on two processors than do binary trees. Since each internal node has an odd number of children, we cannot minimize communication by simply splitting the tree down the middle as with binary trees. Instead, I present upper and lower bounds on communication for computing a complete ternary tree in minimum time. These bounds differ only by an additive constant.

First define the function $N(j)$ to be $\frac{3^{j+1}-1}{2}$. This is the number of nodes in a complete ternary tree of height j.

Theorem 5.1

Let T be the complete ternary tree of height $k \geq 2$, and let h be the largest integer such that $k \geq h + N(h)$. Then T can be scheduled in minimum time with the communication cost at most $k - h + 1$.

Proof

Let

$$L = k - h - N(h) \geq 0$$

By definition of h, $k < h + 1 + N(h+1)$, hence $L < 3^{h+1} + 1$. Now picture the tree T as in Figure 10. Each T_j has height $k - 1 - j$ and the height of each center node $C(j)$ is $k - j$. Let $b = N(h) + \left\lfloor \frac{L}{2} \right\rfloor - 1$, and construct the schedule in the following way. Let processor P

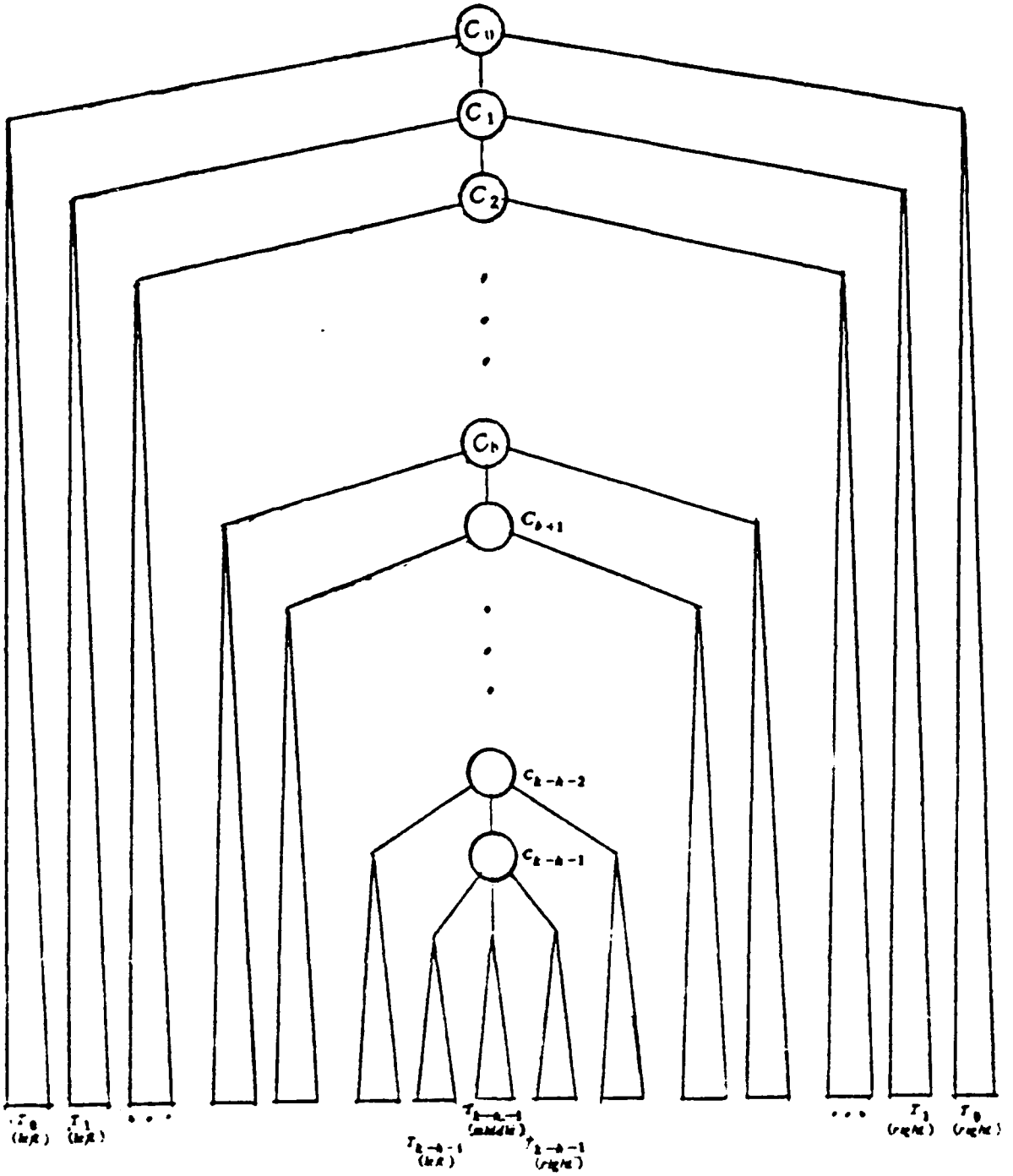


Figure 10:
Complete ternary tree of height k

compute $T_{k-h-1}(middle)$ and subtrees $T_j(left)$ for $0 \leq j \leq k-h-1$. Processor Q computes subtrees $T_j(right)$ for $0 \leq j \leq k-h-1$. Of the center nodes, let Q compute $C(j)$ for $0 \leq j \leq b$ and P compute all others, i.e., $C(j)$ for $b < j \leq k-h$.

For the timing, first notice that $|T_0(R)| > \frac{L}{2}$.

1) Let processors P and Q compute nodes in their subtrees from the inside out, that is, starting with $T_{k-h-1}(u)$, $u \in \{right, left, middle\}$ and working out toward $T_0(u)$.

2) Both processors continue computing these subtrees until processor Q has only $\left\lfloor \frac{L}{2} \right\rfloor$ nodes remaining in those trees. These nodes are all in $T_0(right)$.

3) Processor P will now have $N(h) + \left\lfloor \frac{L}{2} \right\rfloor$ nodes remaining to be computed in $T_0(left)$. None of the C nodes will yet have been computed.

4) In the next $\left\lfloor \frac{L}{2} \right\rfloor$ time units, P computes the bottom C nodes while Q computes the remaining nodes in $T_0(right)$.

5) The remaining uncomputed nodes are all in $T_0(left)$ and in the center chain of C nodes. Processor P has a total of $N(h) + \left\lfloor \frac{L}{2} \right\rfloor$ nodes remaining, while Q has $N(h) + \left\lfloor \frac{L}{2} \right\rfloor + 1$ nodes left to be computed. In the next $N(j) + \left\lfloor \frac{L}{2} \right\rfloor$ time units, let both processors work on their respective nodes. Since none of the nodes assigned to processor Q is a predecessor of any of the nodes assigned to P, and since $C(0)$ is the only node assigned to processor Q that is a successor to those assigned to P, processors P and Q can work simultaneously on these nodes for $N(h) + \left\lfloor \frac{L}{2} \right\rfloor$ time units (i.e., until processor P is finished).

6) Let Q finish the last one (or if L was odd, two) C node(s).

Evidently, since P computes $\left\lfloor \frac{N(k)-1}{2} \right\rfloor$, and Q computes $\left\lfloor \frac{N(k)-1}{2} \right\rfloor$ of the internal nodes, both of the processors are active at all times as much as possible, making time minimal. Now examine the communication cost incurred in this schedule. Because each of the right and left subtrees of all center nodes is computed entirely by one processor, the only communication nodes in the schedule are center nodes. The communication to each $C(j)$ where $0 \leq j \leq b$ is 1, since Q computes both $C(j)$ and two of its children: $C(j+1)$ and the (root of the) subtree $T_j(right)$, whereas P computes the third child of $C(j)$. Communication is also 1 to each $C(j)$, $\forall j$, $b+1 < j \leq b + \left\lfloor \frac{L}{2} \right\rfloor$ since processor P computes $C(j)$ and the (root of) subtree $T_j(left)$ as well as $C(j+1)$ (or $T_j(middle)$ when $j = b + \left\lfloor \frac{L}{2} \right\rfloor$), while Q computes the third child of $C(j)$. The communication to $C(b+1)$, computed by Q, is 2 since Q computes $T_{b+1}(right)$ whereas P computes the other two children of $C(b+1)$. Then the total communication cost is

$$b + \left\lfloor \frac{L}{2} \right\rfloor + 2$$

$$= N(h) - 1 + L - 2 = k - h + 1 \text{ as desired.}$$

Hence, if complete ternary tree T has height $k = h + N(h) + L$, $L < 3^{h-1} + 1$, then T can be scheduled in communication no more than $k - h + 1$.

□

Notice, too, that $h < \log_3 k$, but also, $h \geq \log_3(k-2)$. Then we clearly have a schedule that computes a complete ternary tree in minimum time with communication no more than $k - \lceil \log_3(k-2) \rceil + 1 = k - \log_3 k + 3$.

5.2 A Lower Bound for Communication Cost

Having established an upper bound on communication cost in scheduling complete ternary trees, I now introduce several lemmas and facts about complete ternary trees and their schedules to aid in determining a lower bound.

First, notice that any schedule that minimizes time must partition the non-root nodes into two sets whose sizes differ by at most 1, since both processors must do the same amount of work, and must place the root into one of these sets. Call any partition of the nodes into two such sets a *proper partition*.

5.2.1 Definitions and Notations

Again note that since we are discussing undirected trees, a node's children and descendants are its predecessors (rather than its successors) in the partial ordering described by the tree. Similarly, the ancestors of a node are its successors.

For any complete ternary tree T of height k with proper partition $S = (P, Q)$, define the following (note that all set membership is with respect to S):

For every node v in T, let $G(v) = R \in \{P, Q\}$ such that $v \in R$.

An edge (v, w) is *broken* if $G(v) = P$ and $G(w) = Q$, or vice-versa. Edge (v, w) is *broken to level i* if node w is at level i (v is at level $i + 1$).

Define *comm(S)* to be the number of edges broken in the proper partition S. A proper partition S with a minimal comm(S) is an *optimal partition*.

In a similar vein, *comm*(S, i) denotes the total number of edges broken to all levels at or above level i.

Let $P(i)$ denote the number of nodes in P at level i. Similarly, let $Q(i)$ denote the number of nodes in Q at level i. Then $0 \leq P(i), Q(i) \leq 3^{k-i}$, $1 \leq i \leq k$.

Let $dP(i) = P(i) - Q(i)$ and $dQ(i) = Q(i) - P(i)$ by definition, for every i. both $dP(i)$ and $dQ(i)$ are odd integers.

If $G = P$ then $\bar{G}$ denotes Q, if $G = Q$ then $\bar{G}$ denotes P.

For $G \in \{P, Q\}$, and $0 \leq a \leq 3$, a (G, a)-*receiver* is a node v in G such that exactly a of v's children are in G.

If v is an (G, a)-receiver for some $G \in \{P, Q\}$, $a \in \{0, 1, 2, 3\}$, then the *status* of v is (G, a).

For $a \in \{0, 1, 2, 3\}$, an a-*receiver* is a node that is in the same set as exactly $3 - a$ of its children. Then, an a-receiver is either a (P, a)-receiver or a (Q, a)-receiver.

A Q-*receiver* is a node v such that v is a (Q, a)-receiver for some $1 \leq a \leq 3$. Similarly for a P-*receiver*.

If we denote the nodes of the i'th level of T $v(1), \ldots, v(3^i)$, and say that $v(j)$ is an $a(j)$-receiver for all $1 \leq j \leq 3^i$, then i is an A-*receiver level* if $\sum_{j=1}^{3^i} a(j) = A$.

A node v such that $G(v) = P$ is called a P-*node*; a node w such that $G(w) = Q$ is a Q-*node*.

5.2.2 Elementary Observations

Clearly a lower bound on the number of edges broken in a proper partition S of the non-leaf nodes of a complete ternary tree T is a lower bound on the communication required for scheduling T in minimum time. This is only a lower bound since there may be no schedule with this communication that also satisfies the precedence constraints of the tree.

$$\sum_{i=1}^{\log_3 n} P(i) = \sum_{i=1}^{\log_3 n} Q(i) = \frac{n-1}{2} \quad \text{where } n = 3^k, k \in \{1, 2, \ldots\}$$

$\forall i,\ 1 \leq i \leq k,\ dP(i) = -dQ(i)$.

$\sum_{i=1}^{k} dP(i) \leq 1$.

For all k, the complete ternary tree T_k of height k has $N(k)$ nodes, of which 3^k are leaves. Thus, more than half of the nodes of T_k are leaves, so $P(k), Q(k) > 0$. Furthermore, if $k > 2$, then $P(k), Q(k) \geq 4$.

5.2.3 Schedule Transformations

First I shall force the structure of an optimal partition of a complete ternary tree T of height $k \geq 3$ into a strict form, thereby facilitating analysis. Toward this end I introduce three transformations **Remove3, Remove2**, and **Elim2**, on such partitions. Although **Remove3** and **Remove2** are defined in terms of P-receivers, the transformations are symmetrical for Q-receivers.

1) Remove3(v)

Suppose $S = (P, Q)$ has at least one $(P, 3)$-receiver v. Then in the following manner we transform S to $S' = (P', Q')$ in which

i) v is a $(Q', 0)$-receiver, and

ii) $\text{comm}(S') < \text{comm}(S)$.

Pick a leaf w in Q such that w is not a child of v. Then let

$P' = (P - \{v\}) \cup \{w\}$.
$Q' = (Q - \{w\}) \cup \{v\}$.

Since v is a $(P, 3)$-receiver, all of v's children were Q-nodes. Then all of v's children are Q-nodes, and v is a $(Q', 0)$-receiver.

Now consider $\text{comm}(S)$. The three edges from v to v's children are no longer broken in S'. If edges $(par(v), v)$ and $(par(w), w)$ were not broken in S, then they are broken in S'.

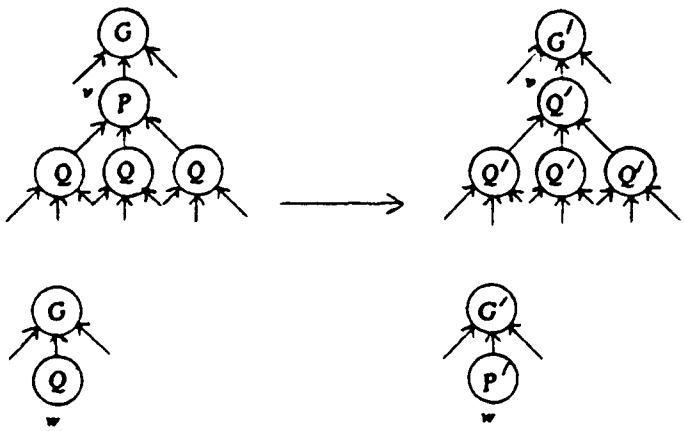


Figure 11:
Remove3

since v and w have been switched to Q' and P' respectively. This could introduce at most two new broken edges. So

$$comm(S') \leq comm(S) - 3 + 2 < comm(S).$$

2) Remove2(v)

If S has at least one $(P, 2)$-receiver w whose parent x is in Q, then the following procedure transforms $S = (P, Q)$ to $S' = (P', Q')$ such that

i) w is a $(Q', 1)$-receiver, and

ii) $comm(S') \leq comm(S)$

Again, pick a leaf w in Q such that w is not a child of v. Then let

$$P' = (P - \{w\}) \cup \{x\},$$
$$Q' = (Q - \{x\}) \cup \{w\}.$$

In this case, v is now a $(Q', 1)$-receiver since one of its children is in P' whereas the others are in Q'. Also, x receives one fewer broken arc. At most one new arc, that from w to par(w), may be broken in S'. Consequently,

$$\text{comm}(S') \leq \text{comm}(S) - 2 + 1 < \text{comm}(S).$$

Note that both Remove3 and Remove2 cause the total number of broken edges to decrease. Consequently, no optimal partition can have either of the situations allowing Remove3 or Remove2 to be executed.

3) **Elim2**(v, x)

If $S = (P, Q)$ is a partition in which

a) v is a $(P, 2)$-receiver,

b) w is a $(Q, 2)$-receiver, and

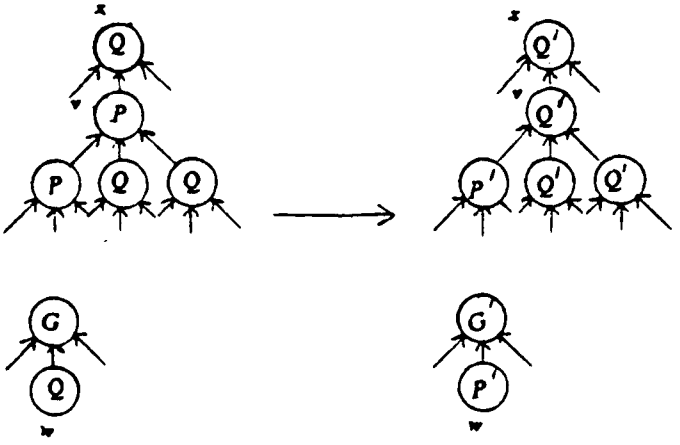


Figure 12:
Remove2

c) $par(v) \in P$; $par(x) \in Q$.

then we transform S to $S' = (P', Q')$ in which

i) v is a $(Q', 1)$-receiver,

ii) x is a $(P', 1)$-receiver, and

iii) $comm(S') = comm(S)$.

in the following manner.

Let

$P' = (P - \{v\}) \cup \{x\}$.
$Q' = (Q - \{x\}) \cup \{v\}$.

The only edges affected are those into and out of v and x. Since only one of v's children is in P, only one arc to v is broken in S'. Similarly, one of x's incoming edges is broken in S'. Both $(par(v), v)$ and $(par(x), x)$ are broken in S'. Then exactly 4 of the 8 edges surrounding

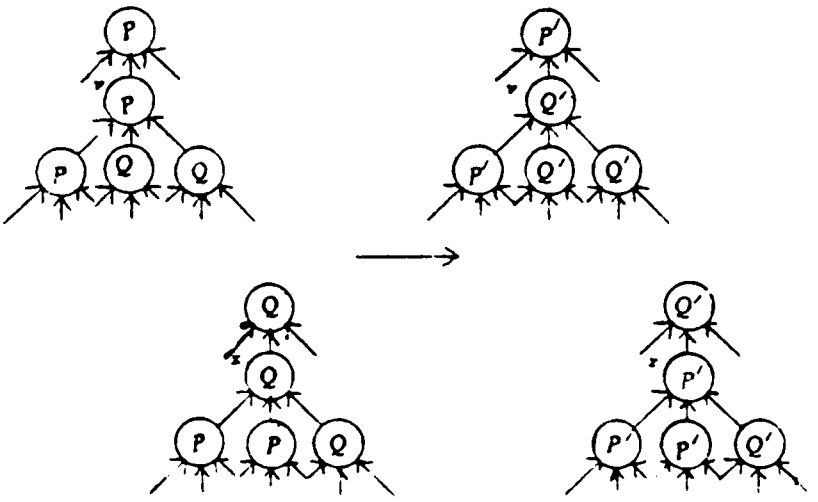


Figure 13:
Elim2

v and x are broken in S'. In S, each of v and x had 2 broken edges surrounding it which again totals 4. Since no other edges were affected by the transformation,

$$comm(S') = comm(S)$$

5.2.4 Lemmas and Corollaries

Lemma 5.1 (The Left Property)

We can assume, without loss of generality, that in an optimal partition of a complete ternary tree no Q-node lies to the left of any P-node at any level.

Proof

Consider a complete ternary tree T with optimal partition $S = (P, Q)$. For a level i, let $P(i) = m$, $P(i+1) = n$. Then the number of edges to level i that are broken in S must be at least $|3m - n|$. That is, since the m P-nodes at level i have a total of $3m$ children at level $i + 1$, if $n < 3m$, then at least $3m - n$ of those children must be in Q; similarly, if $n > 3m$, then at least $n - 3m$ P-nodes on level $i + 1$ must have parents in Q. Then any partition must have at least

$$\sum_{j=1}^{i-1} |3P(j) - P(j+1)|$$

broken edges. This minimum can be achieved by putting the $P(i)$ leftmost nodes in P and the $Q(i)$ rightmost nodes in Q at each level i.

□

Corollary 5.1 (The Simple Property)

If the Left Property holds in a partition $S = (P, Q)$ of a complete ternary tree T, then

i) No level has both a P-receiver and a Q-receiver.

ii) No level has more than one b-receiver where $b \in \{1, 2\}$.

Furthermore, no level has both a 1-receiver and a 2-receiver.

Proof

Let T be a complete ternary tree with partition $S = (P, Q)$ having the left property. Then let some level i have a P-receiver. Examine the rightmost P-receiver v. Since v is a P-receiver, at least one of its children w must be a Q-node. By the Left Property, all nodes to the right of w on level $i + 1$ are in Q. Since all Q-nodes on level i are to the right of v, all of their children must also be Q-nodes; hence none are Q-receivers, and level i has no Q-receiver. Ergo, no level has both a P-receiver and a Q-receiver.

To demonstrate (ii), let v be a (P, b)-receiver on some level i, $b \in \{1, 2\}$. Let w be the leftmost child of v and x be the rightmost child of v. Since $1 \leq b \leq 2$, w is a P-node, and x is a Q-node. By the Left Property, all nodes to the left of w on level $i + 1$ are P-nodes; consequently no node to the left of v on level i can be a P-receiver. All nodes to the right of x on level $i + 1$ are Q-nodes; thus, all of the children of each node to the right of v on level i are in Q. Therefore any P-nodes to the right of v on level i are $(P, 3)$-receivers, and level i has no other $(P, 1)$- or $(P, 2)$-receivers. The proof for $(Q, 1)$- and $(Q, 2)$-receivers strictly parallels the last argument.

□

I would like to force an optimal partition $S = (P, Q)$ of a complete ternary tree of height at least 3 that has the Left Property into a partition $S' = (P', Q')$ where

i. $\text{comm}(S') = \text{comm}(S)$,

ii. S' maintains both the simple and Left Properties,

iii. If S has a (P, b)-receiver, then S' has no $(Q, 2)$-receivers.

We will also show that an optimal partition has no 3-receivers. If this is possible, then S' is an optimal

partition has this form. To force the partition into the above form, I use the following algorithm.

```
1     Procedure FormatPartition

2     begin
3     while there are at least one (P, 2) and one (Q, 2)-receiver
          above level k − 1 do

4         begin
5             v := the lowest (P, 2)-receiver above level k − 1
                  (that is, the one closest to the leaves)
6             x := the lowest (Q, 2)-receiver above level k − 1
7             Elim2(v, x); Replace S with S'
8         end

9       S' := S
10    end.
```

Lemma 5.2

Given an optimal partition $S = (P, Q)$ of a complete ternary tree T as input, FormatPartition converts S to $S' = (P', Q')$ as described in (i) through (iii) above.

Proof

First, by Lemma 5.1, we can assume that the Left Property holds for S. The only changes made to S are **Elim2** transformations. We know that comm(S) is not affected by **Elim2**(v, x), so (i) is obvious. I claim that the Left Property still holds after each execution of **Elim2R**(v, x). This must be true if it holds before the execution. Consider node v. Node v is a (P, 2)-receiver. Thus exactly two of the children of v are Q-nodes. Then all nodes to the right of rc(v) are Q-nodes, and all nodes to the left of lc(v) are P-nodes. Since the partition is optimal it has no 3-receivers, so all nodes to the right of v must also be Q-nodes, and all nodes to its left are P-nodes. **Elim2**(v, x) converts v to a Q-node. Since no P-nodes were to its right, and no other node on that level is changed, all nodes to the right are still Q-nodes, and all

nodes to its left are P-nodes. Therefore, the Left Property still holds on that level. The argument is symmetric for the level of x. Hence, after each execution of **Elim2**(v, x), the Left and, by Corollary 5.1, Simple Properties hold.

In order to show that FormatPartition terminates I show that the number of times the test at line 3 is executed is bounded by the height of the tree. The procedure is repeated only if both (P, 2)-receivers and (Q, 2)-receivers are present in the partition at the time of the test. If so, the 2-receiver w closest to the leaves of T is eliminated by **Elim2**(v, x). All nodes whose status is changed are above the level of w. Then after **Elim2**(v, x) (v or x = w) is executed, the lowest 2-receiver in the partition is higher in the tree than before, and none of the other transformations introduce any 2-receivers at any lower level. After the i th time through the **while** loop, the lowest 2-receiver is no lower than level $k - i$. Consequently, since the **while** loop terminates if no more than one 2-receiver remains in the partition, the number of loop iterations is at most $k - 1$, and therefore it must terminate.

Upon termination, properties (i) and (ii) of Lemma 5.2 hold. Furthermore, property (iii) is the termination condition of the procedure, so property (iii) also holds. Consequently, an optimal partition can be forced into the form stated above.

□

At this point some observations about the effect of different kinds of receivers in an optimal partition are helpful. For Lemmas 5.3 through 5.6 and Corollaries 5.2 and 5.3, T is a complete ternary tree with proper partition $S = (P, Q)$ of the form dictated in Lemma 5.2.

Lemma 5.3

If S has no (Q, 2)-receiver at level h, h > 0, and $dQ(h) \geq 1$, then $dQ(h-1) \geq dQ(h)$ (similarly for P).

Proof

If level h has no Q-receivers at all, then each child of every Q-node at level h is a Q-node. Then

$$Q(h+1) \geq 3(Q(h)).$$

Consequently,

$$P(h+1) \leq 3(P(h)).$$

Then

$$dQ(h+1) \geq 3dQ(h) > dQ(h).$$

If level h has a $(Q, 1)$-receiver then

$$Q(h+1) = 3Q(h) - 1$$

$$P(h+1) = 3P(h) + 1.$$

Then

$$dQ(h+1) = 3Q(h) - 3P(h) - 2$$

$$= 3dQ(h) - 2.$$

Since $dQ(h) \geq 1$,

$$dQ(h+1) \geq 3dQ(h) - 2dQ(h) = dQ(h).$$

By hypothesis, these are the only two cases possible.

□

Corollary 5.2

If S and T are as described in Lemma 5.3, and there are levels h and g such that $h \leq g \leq k-1$, and $dQ(h) \geq 1$, and S has no $(Q, 2)$-receivers at any level i, $h \leq i \leq g$, then for all levels i, $h \leq i \leq g$, $dQ(g) \geq dQ(i)$.

Proof

By Lemma 5.3 applied $g - h - i$ times, $dQ(h) \leqslant dQ(h+1) \leqslant \quad \leqslant dQ(g)$

□

Lemma 5.4

If there is a level j in T such that $dP(j) > 0$ while $dP(j-1) < 0$, then

a) $dP(j) = 1$,

b) $dQ(j-1) = 1$, and

c) level $j-1$ has a $(Q, 2)$-receiver.

Proof

By definition, level $j-1$ can have no 3-receivers and can have at most one 1- or 2-receiver. Since $dP(j-1) < 0$ but $dP(j+1) > 0$, $j-1$ must have a Q-receiver v. If v is a $(Q, 1)$-receiver, then

$$\begin{aligned} dP(j) &= (3P(j-1)+1) - (3Q(j-1)-1) \\ &= 3dP(j-1) + 2 \\ &\leqslant -3 + 1 = -1. \end{aligned}$$

Thus, v must be a $(Q, 2)$-receiver. Then

$$\begin{aligned} dP(j) &= (3P(j-1)+2) - (3Q(j-1)-2) \\ &= 3dP(j-1) + 4 \\ &\leqslant -3 + 4 = 1. \end{aligned}$$

Hence, since $dP(j) > 0$, $dP(j) = 1$, and $3dP(j-1) = -3$, so $dP(j-1) = -1$ or, equivalently, $dQ(j-1) = 1$.

Lemma 5.5

If S has no $(Q, 2)$-receivers above level $k-1$, then for any level l where $1 \leqslant l \leqslant k-1$, $dP(l) = 1$ iff

a) $dP(l-1) = 1$, and

b) S has a $(P, 1)$-receiver at level $l-1$.

Proof

The proof of Lemma 5.5 has two parts. First I show sufficiency of (a) and (b).

If $dP(l-1) = 1$ and level $l-1$ has a $(P, 1)$-receiver, then

$P(l) = 3P(l-1) - 1.$

$Q(l) = 3Q(l-1) + 1$ implying $dP(l) = 3dP(l-1) - 2 = 1.$

To show necessity of (a) and (b) assume that $dP(l) = 1$. Then since S has no $(Q, 2)$-receivers above level l, $dP(l-1) \geqslant 1$; otherwise, if $dP(l-1) < 1$, then $dQ(l-1) \geqslant 1$, hence $dQ(l) \geqslant 1$ (by Lemma 5.3) and thus $dP(l) \leqslant -1$ contradicting the assumption $dP(l) = 1$.

Suppose, then, that $dP(l-1) > 1$. Then

$$\begin{aligned} P(l) &\geqslant 3P(l-1) - 2 \\ &\geqslant 3(Q(l-1) + 2) - 2 \\ &= 3(Q(l-1) + 4. \end{aligned}$$

$$Q(l) \leqslant 3Q(l-1) + 2.$$

Then

$dP(l) \geqslant 4 - 2 = 2.$

hence by the assumption $dP(l) = 1$, condition (a) is necessary. Then

$P(l) = 3P(l-1) + \epsilon$; $Q(l) = 3Q(l-1) - \epsilon$ for some $-2 \leqslant \epsilon \leqslant 2$

$$dP(l) = 3P(l-1) - 3Q(l-1) + 2\epsilon$$
$$= 3dP(l-1) + 2\epsilon$$
$$= 3 + 2\epsilon.$$

Then

$2\epsilon = -2$ implying $\epsilon = -1$.

Since $P(l) = 3P(l-1) - 1$, level $l-1$ must have a $(P, 1)$-receiver on it.

□

Corollary 5.3

If there is a level j in T such that $dP(j) > 0$ while $dP(j-1) < 0$, then there are no $(Q, 2)$-receivers above level $j-1$ in T.

Proof

By Lemma 5.4, there is a $(Q, 2)$-receiver on level $j-1$. Consequently, by definition, there are no $(P, \geq 2)$-receivers in the tree. Furthermore, since $dQ(j-1) = 1$, by applying Lemma 5.5 $j-1$ times, we find

$$dQ(j-2) = dQ(j-3) = \quad = dQ(0) = 1.$$

Furthermore, there is a $(Q, 1)$-receiver on each of these levels; thus no level above level j has a $(Q, 2)$-receiver.

□

Recall the definition of $N(j)$ (see section 5.1).

Lemma 5.6

Suppose T is optimal, and there is some level $h < \ell$ such that

1. $dQ(j) \geq 1$
2. there are m 0-receiver levels between the leaves below level h.

of which r' is the level of the children of the lowest one, and

3) S has no $(Q, 2)$-receivers above level $k - 1$.

Then

a) $dQ(r) \geq 3^m$

b) $\sum_{i=r'} dQ(i) \geq N(m)$

Proof

The proof is by induction on m.

Suppose $m = 1$. Since S has no $(Q, 2)$-receivers, by Corollary 5.2, $dQ(r-1) \geq 1$. By definition of r, level $r - 1$ is a 0-receiver level so $dQ(r) = 3dQ(r-1) > 3 = 3^1$ for condition (a). Since

$dQ(h) \geq 1$,

$\sum_{i=r'} dQ(i) \geq 1 + 3 = 4 = N(1) = N(m)$ which is condition (b).

Now assume that the lemma is true for all $m < n$. With $m = n$, r is the level of the children of the n th 0-receiver level, let r' be the level of the children of the $n-1$st 0-receiver level. By the induction hypothesis,

$dQ(r') \geq 3^{n-1}$

and

$\sum_{i=r'} dQ(i) \geq N(n-1)$

Since $r' \leq r - 1$, by Corollary 5.2, with $h = r'$, $g = r - 1$, $dQ(r-1) \geq dQ(r')$. By definition, level $r - 1$ is a 0-receiver level, so

$dQ(r) = 3dQ(r-1)$

$$\geq 3dQ(r')$$

$$\geq 3(3^{n-1}) = 3^n .$$

Then

$$\sum_{i=h}^{r} dQ(i) \geq \sum_{i=h}^{r'} dQ(i) + dQ(r) \geq N(n-1) + 3^n = N(n).$$

□

5.2.5 The Lower Bound

I now establish the lower bound for the number of edges broken in a proper partition of a complete ternary tree of height k.

Theorem 5.2

Every proper partition $S = (P, Q)$ of a complete ternary tree T of height $k > 1$ has more than $k - \log_3 k + 1$ broken edges.

Proof

First consider $k = 2$. To make a proper partition of the complete ternary tree of height 2, at least three ($> 2 - \log_3 2 + 1$) edges must be broken. We see this because, since T_2 has 12 non-root nodes, each of the sets P and Q must have 6 non-root nodes in it. Either P or Q must have more of the nodes at level 1 in it than does the other. Without loss of generality, assume that P has more. If all of the nodes at level 1 are in P then six nodes at level 2 must be in Q, which causes six edges to be broken. If, however, two nodes at level 1 are in P, then at least one edge to the root must be broken. Furthermore, to get 5 nodes at level 2 in Q, at least two edges to level 1 must be broken; thus at least three edges are broken.

For $k \geq 3$, I will prove Theorem 5.2 by contradiction. Let $S = (P, Q)$ be an optimal partition of T, and suppose $\text{comm}(S) \leq k - \log_3 k + 1$. Note that $\text{comm}(S) = \text{comm}(S_{k-1})$ since S_{k-1} can have no edges broken to the leaves (since there are no edges to the leaves).

By Lemma 5.2, we can assume that S has at most one 1- or 2-receiver node at any level above $k-1$ and no 3-receivers anywhere above level $k-1$. Because T is a complete ternary tree, each level has an odd number of nodes. Thus, for all $1 \leqslant i \leqslant k-1$, $P(i) \neq Q(i)$. Without loss of generality, let $P(1) > Q(1)$.

If there is no level j such that $Q(j) > P(j)$ then $\sum_{i=1}^{k} dP(i) \geqslant k > 1$, hence the partition is not proper. Consequently, there must be at least one level j such that $P(j) > Q(j)$ but $P(j+1) < Q(j+1)$. Call any such level a *switch-to-Q level*. If level $k-1$ is the only switch-to-Q level in T, let $j = k-1$. Otherwise, let j be the switch-to-Q level furthest down in the tree other than $k-1$. Now we have two cases.

Case I $j < k-1$

By Lemma 5.4, $dP(j) = 1$, and S has a $(P, 2)$-receiver at level j in T. Consequently, by Lemma 5.2, S has no $(Q, 2)$-receivers anywhere in T except, perhaps, at level $k-1$. Furthermore, by Corollary 5.3, S has no $(P, 2)$-receivers at any level above level j; hence j is the highest switch-to-Q level. In addition, by Lemma 5.4, S has no switch-to-P levels above $k-1$ since a switch-to-P level requires a $(Q, 2)$-receiver; consequently, there is no level i above j such that $Q(i) > P(i)$. By Lemma 5.5, since $dP(j) = 1$, $dP(j-1) = 1$ and level $j-1$ has a $(P, 1)$-receiver. By applying Lemma 5.5 $j-1$ times, we see that for all $1 \leqslant i < j$, $dP(i) = 1$ and level i has a $(P, 1)$-receiver, and level 0 must be at least a 1-receiver level. Level j has a $(P, 2)$-receiver, so $\mathrm{comm}(S, j) = j+2$ and

$$\sum_{i=1}^{j} P(i) = \sum_{i=1}^{j} Q(i) + j.$$

Since S must be a proper partition,

$$\sum_{i=j+1}^{k} P(i) = \sum_{i=j+1}^{k} Q(i) - j + \epsilon$$

where $\epsilon \in \{-1, 0, 1\}$.

If at least one edge on each level below j is broken, then $\mathrm{comm}(S, k-1) \geq k+1$. For $\mathrm{comm}(S) \leq k - \log_3 k + 1$, at least

$$k + 1 - (k - \log_3 k + 1) = \log_3 k$$

levels between j and k (exclusive) must be 0-receiver levels. Clearly, all conditions for Lemma 5.6 with $h = j + 1$ and $m = \log_3 k$ are met. Then by Lemma 5.6,

$$\sum_{i=j+1}^{r} dQ(i) \geq N(\log_3 k)$$

$$= \frac{3^{\log_3 k + 1} - 1}{2}$$

$$= \frac{3k - 1}{2}$$

where r is the level of the children of the lowest 0-receiver level (not including the leaves). If $r = k$, let $l = k$; otherwise, $l = k - 1$. Then, again since S has no $(Q, 2)$-receivers above level l, by corollary 5.2, $dQ(i) \geq dQ(r)$ for all i, $r \leq i \leq l$. Thus,

$$\sum_{i=j+1}^{l} dQ(i) \geq \frac{3k - 1}{2}.$$

Two possible subcases arise at this juncture.

Case I.1 $Q(k) > P(k)$

In this case,

$$\sum_{i=j+1}^{r} dQ(i) \geq \frac{3k - 1}{2}.$$

Since S has at least $\log_3 k + 1$ 0-receiver levels (including the leaves), T has at least that many levels below j. Then

$$k \geq j - \log_3 k - 1.$$

Clearly, then

$$\frac{3k - 1}{2} \leq \sum_{i=j+1}^{r} dQ(i)$$

$$= \sum_{i=1}^{j} dP(i) + \epsilon$$

$$= j + \epsilon, \quad \epsilon \in \{-1, 0, 1\}.$$

Furthermore,

$$k > 1 \text{ implies } k - \log_3 k < \frac{3k - 1}{2}.$$

and

$$k \geqslant j + \log_3 k + 1 \text{ implies } j \leqslant k - \log_3 k - 1.$$

so

$$j + \epsilon \leqslant k - \log_3 k .$$

Then

$$k - \log_3 k \geqslant j + \epsilon = \sum_{i=j+1}^{k} dQ(i)$$

$$\geqslant \frac{3k - 1}{2} > k - \log_3 k .$$

or

$$k - \log_3 k > k - \log_3 k$$

which is a contradiction. Thus, in this case $\text{comm}(S) > k - \log_3 k + 1$.

Case I.2 $P(k) > Q(k)$

Since $j < k - 1$, we still know that $dQ(k - 1) \geqslant 3^{\log_3 k} = k$. Since $P(k) > Q(k)$, at least $\left\lfloor \frac{3^k - 1}{2} \right\rfloor$ of the nodes at level $k - 1$ have children in P.

Concerning the nodes at level $k - 1$, we know that

$$3^{k-1} = 2P(k - 1) + dQ(k - 1);$$

i.e the total number of nodes at level $k - 1$ is

$$P(k-1)+(Q(k-1)-P(k-1))+P(k-1)$$

Then

$$\left|\frac{3^{k}-1}{2}\right| = P(k-1)+\left|\frac{dQ(k-1)}{2}\right|.$$

So at least $\left|\frac{dQ(k-1)}{2}\right|$ Q-nodes on level $k-1$ have children in P hence at least

$$\left|\frac{dQ(k-1)}{2}\times 3\right| = dQ(k-1)+\left|\frac{dQ(k-1)}{2}\right| \geq k+1$$

edges are broken. Then the overall partition has

$$\text{comm}(S) \geq j+1+k+1 > k-\log_3 k+1$$

which is a contradiction. Therefore, in this case, $\text{comm}(S) > k-\log_3 k+1$.

Case II $j = k-1$ i.e., for all $1 \leq i < k$, $P(i) > Q(i)$; $P(k) < Q(k)$

By Lemma 5.4, if $P(k) < Q(k)$ while $P(k-1) > Q(k-1)$, at least 2 edges to level $k-1$ must be broken. If at least one edge is broken to each level above $k-1$ then $\text{comm}(S) \geq k+1$. To get $\text{comm}(S) \leq k-\log_3 k+1$, at least $k+1-(k-\log_3 k+1)$ levels above $k-1$ must be 0-receiver levels. Since $P(i) > Q(i)$ for all $1 \leq i < k$, by Lemma 5.6,

$$dP(k-1) \geq 3^{\log_3 k} = k,$$

and

$$\sum_{i=1}^{k-1} dP(i) \geq \frac{3^{\log_3 k+1}-1}{2} = \frac{3k-1}{2}.$$

Because S is a proper partition,

$$dQ(k) = \sum_{i=1}^{k-1} dP(i) + \epsilon$$

Particularly, $dQ(k) \geq 1$. Then by the argument given in Case I.2, at least

$$\left|\frac{3dQ(k-1)}{2}\right| \geq \left|\frac{3k-1}{2}\right| \geq k-\log_3 k+1$$

edges are broken to level $k - 1$. Since at least as many edges are broken in the entire partition as are broken at the lowest level, this cannot give a partition with $\text{comm}(S) \leq k - \log_3 k + 1$.

□

5.3 Comparison of Upper and Lower Bounds

Unsurprisingly, the upper and lower bounds presented in Sections 5.1 and 5.2 are quite tight. They are within one of each other for all values of k, and correspond on many.

Corollary 5.4

If h is as defined as in Section 5.1, i.e., h is the largest integer such that $k \geq h + N(h)$, the lower bound for communication cost is achievable for all trees of height k such that $k \geq 3^{h+1}$ using the schedule presented.

Proof

The proof is algebraic. For T of height $k = h + N(h) + L$, the schedule presented in Section 5.1 has communication cost $k - h + 1$. If the lower bound is achieved, then

$$k - \log_3 k + 1 < k - h + 1 \leq k - \log_3 k + 2.$$

Thus,

$$\log_3 k > h \geq \log_3 k - 1,$$

or $k > 3^h$, and $3^{h+1} \geq k$. For this to be so,

$$L \geq \frac{3^{h+1} + 1}{2} - h.$$

For h to be within the right range, L must be no more than $3^{h+1} - 1 + h$. The value computed above lies well within this range. Thus, for

$$2h + \frac{3^{h+2} - 1}{2} > k \geq 3^{h+1},$$

the lower bound on communication is achieveble. This is not surprising since the schedule presented in Section 5.1 follows the format of the optimal partition presented in Section 5.2 very closely at least as far down as the switch-to-Q level, which is as far as I define it.

□

CHAPTER 6

CONCLUSIONS AND OPEN PROBLEMS

Scheduling dags on many processors to minimize completion time has long been known to be a difficult problem. In some restricted cases, however, the problem is not so bleak. When we try to schedule a general dag on two processors, or a tree on any number of processors, we can do so in time polynomial in the size of the graph.

Knowing this, it is reasonable to wonder how difficult the problem becomes if we introduce a new constraint, that of minimizing communication cost, to the problem. Although scheduling a general dag on two processors in minimum time can be done in polynomial time, I have shown that when a communication cost constraint is added, the problem again becomes NP-complete. Afrati et al. [Afrati 1985] show that scheduling a tree on an arbitrary number of processors is also an NP-complete problem.

The difficulty of the problem of scheduling a tree on two processors in minimum tine and communication cost cannot be directly inferred from the previous results, and it remains open. For complete binary and ternary trees, however, I have determined upper and lower bounds on the communication cost of computation in minimum time.

First, a complete binary tree can be computed in minimum time on two processors with communication cost 1. This is also the minimum communication that must be incurred. Second, I show that a complete ternary tree of height k can be scheduled in minimum time with communication cost no more than $k - h + 1$ where h is defined to be the largest integer such that $k \geq h + \frac{3^{h+1} - 1}{2}$. As a lower bound, I show that communication cost greater than $k - \log_3 k + 1$ is required for a minimum time schedule. Comparing these bounds, we can see that a lower bound is achieavable for an infinite number of trees using the algorithm I

presented.

Several questions remain to be solved, primarily concerning scheduling trees.

1) Is the problem of scheduling an in-directed tree on two processors in minimum time and communication NP-complete?

2) If so, what happens if we further restrict the tree to a binary trees? Is it easier to schedule a binary tree in minimum time and communication?

3) How difficult is it to schedule an out-directed tree on an arbitrary number of processors in minimum time and communication cost (using Definition 2 of communication cost)?

4) Again, if the problem is NP-complete, how difficult is it to schedule an out-directed tree on two processors? How difficult for an out-directed binary tree?

REFERENCES

Afrati, F., Papadimitriou, C. H., and Papageorgiou, G. [Afrati 1985], "Scheduling Dags to Minimize Time and Communication," Extended Abstract.

Aggarwal, A. and Chandra, A. K. [Aggarwal 1985], "Communication Complexity of PRAMS," Extended Abstract.

Chu, W. W., Holloway, L. J., Lan, M-T., and Efe, K. [Chu 1980], "Task Allocation in Distributed Data Processing," *Computer*, November 1980, pp. 57-69.

Coffman, E. G. Jr. and Graham, R. L. [Coffman 1972], "Optimal Scheduling for Two-Processor Systems," *Acta Informatica*, Vol. 1, 1972, pp. 200-213.

Gabow, H. N. [Gabow 1982], "An Almost-Linear Algorithm for Two-Processor Scheduling," *Journal of the Association for Computing Machinery*, Vol. 29, No. 3, July 1982, pp. 766-780.

Garey, M. R. and Johnson, D. S. [Garey 1975], "Complexity Results for Multiprocessor Scheduling Under Resource Constraints," *SIAM J. Comput.*, Vol. 4, No. 4, December 1975, pp. 397-411.

Garey, M. R. and Johnson, D. S. [Garey 1979], *Computers and Intractability*, W. H. Freeman and Co., New York, N. Y., 1979.

Garey, M. R., Johnson, D. S., and Stockmeyer, L. [Garey 1976], "Some Simplified NP-Complete Graph Problems," *Theoretical Computer Science*, Vol. 1, 1976, pp. 237-267.

Hu, T. C. [Hu 1961], "Parallel Sequencing and Assembly Line Problems," *Operations Res.* Vol. 9, 1961, pp. 841-848.

Lo, V. M. [Lo 1983], "Task Assignment in Distributed Systems," Doctoral Dissertation, U. of Illinois at U-C., Oct., 1983.

Papadimitriou, C. H. and Sipser, M. [Papadimitriou 1984a], "Communication Complexity," *J. of Computer and System Sciences*, Vol. 28, 1984, pp. 260-269.

Papadimitriou, C. H. and Tsitsiklis, J. [Papadimitriou 1982], "On the Complexity of Designing Distributed Protocols," *Information and Control*, Vol. 53, 1982, pp. 211-218.

Papadimitriou, C. H. and Ullman, J. D. [Papadimitriou 1984b], "A Communication Time Tradeoff," FOCS 1984, pp. 84-88.

Stone, H. S. [Stone 1977], "Multiprocessor Scheduling with the Aid of Network Flow Algorithms," *IEEE Trans. Software Eng.*, Vol. SE-3, No. 1, Jan. 1977, pp. 85-93.

Tompa, M. [Tompa 1980], "Time-Space Tradeoffs for Computing Functions, Using Connectivity Properties of Their Circuits," *J. of Computer and System Sciences*, Vol. 20, 1980, pp. 118-132.

Ullman, J. D. [Ullman 1975], "NP-Complete Scheduling Problems," *J. of Computer and System Sciences*, Vol. 10, 1975, pp. 384-393.

END

3-87

DTIC